Merge DICOM Toolkit™

.NET/C# User's Manual

v.4.6.0

Merge Healthcare
200 E. Randolph Street,
Chicago, IL USA, 60601
877.44.MERGE

Copyright 2012 Merge Healthcare Incorporated
Unauthorized use, reproduction, or disclosure is prohibited.

This document has been prepared by Merge Healthcare Incorporated, for its customers. The content of this document is confidential. It may be reproduced only with written permission from Merge Healthcare. Specifications contained herein are subject to change, and these changes will be reported in subsequent revisions or editions.

Merge Healthcare® is a registered trademark of Merge Healthcare Incorporated.

DICOM is a registered trademark of National Electrical Manufacturers Association (NEMA). *Merge DICOM Toolkit*™ is a trademark of Merge Healthcare. The names of other products mentioned in this document may be the trademarks or registered trademarks of their respective companies.

For assistance, please contact Merge Healthcare Customer Support:

- In North America, call toll free 1-800-668-7990, then select option 2
- International, call Merge Healthcare (in Canada) +1-905-672-7990, then select option 2
- Email mdtsupport@merge.com

Part	Date	Revision	Description
COM-390	May 2012	1.0	Updated bi-annually

Contents

Contents iii

Overview 1

The DICOM Standard 1

The Merge DICOM Toolkit 5

Development Platform Requirements 6

Assembly Structure 6
Merge DICOM C/C++ Toolkit Dynamic Library 7
Binary Message Information and Data Dictionary Files 8
Sample Applications 8
Merge DICOM Message Database Manual and Tools 8

Documentation Roadmap 9

Conventions 9

Understanding DICOM 10

General Concepts 10
Application Entities 10
Services and Meta Services 10
Information Model 15

Networking 16
Commands 16
Association Negotiation 17

Messages 18
DICOM Data Dictionary 19
Message Handling 20
Private Attributes 21

Media Interchange 22
DICOM Files 22
File Sets 28
The DICOMDIR 28
File Management Roles and Services 31

Conformance 33

Using the Merge DICOM Toolkit 34

Configuration 34

Initialization File 35
Application Profile 35
System Profile 42
Service Profile 44

Message Logging 44

Utility Programs 45
mc3comp 46
mc3conv 46
mc3echo 47
mc3list 47
mc3valid 48
mc3file 49

Developing DICOM Applications 51

Library Import 53

Library Constants 53

Exception Handling 53

Library Initialization 56

Releasing the library 57

Getting the Assembly Version 57

Releasing Native Memory 57

Using the Merge DICOM log file 58

Capturing Log Messages in Your Application 58

Registering Your Application 59
MCapplication objects can be disposed 59
The Application Entity (AE) Title 59

Association Management (Network Only) 60
Preparing a Proposed Context List 60
Using a Pre-configured Proposed Context List 60
Creating Your Own Proposed Context List 60
Using a Pre-configured Transfer Syntax List 60
Creating Your Own Transfer Syntax List 61
Creating Your Own Proposed Context List 61
MCproposedContext properties 61
MCproposedContextList properties 62
MCresultContext properties 63
MCtransferSyntax properties 63
MCtransferSyntaxList properties 64
Using Extended Negotiation Information 64
Starting an Association Requester 65

Starting an Association Acceptor 66
Accepting or Rejecting the Association 67
Negotiated Transfer Syntaxes 69
Merge DICOM Message Classes 71
Association Message Handling 71
Releasing or Aborting the Association 72
Association Properties 73
Application Context Name 73
TCP/IP Listen Port 73
MCapplication object of the local AE 73
Application Entity Title 74
Implementation Class UID and Implementation Version 74
Maximum PDU Sizes 74
The Proposed Context List 74
The Read Timeout Value 74
The Remote Host's Name and Address 75
Association Role 75
Association State 75

Using the MCsopClass class 75

Using the MCvr class 76

Using the MCtag class 77
Constructing non-private tags 78
Constructing private tags 78

Using the MCdataElement class 78
Constructing standard data elements 79
Constructing non-standard data elements 79

Working With Attribute Sets 80
Constructing Message Objects 80
Construct a message using a pre-populated data set: 80
Construct a message with an empty data set: 81
Construct a message using an existing data set: 81
Convert an MCfile object 82
MCdimseMessage Properties 82
Transfer Syntax Used 82
Contained attribute sets 82
The service and command used by the message 82
MCdimseMessage Command Set Properties 82
Constructing File Objects 84
Construct with a pre-populated data set: 84
Construct with an empty data set: 85
Convert an MCdimseMessage object to an MCfile object 85
Setting data set values 85
Specifying the file name 86
Constructing Item Objects 86
Get/Set item name 86
Constructing MCdataSet Objects 86
Retrieving Contained Attribute Sets 87
Using the MCattribute class 87
Adding Attributes to an Attribute Set 88

Using the MCattributeSet indexer to access MCattribute instances 88
Removing Attributes from an Attribute Set 89
Attribute Properties 89
Assigning Attribute Values from MCattribute 89
Assigning Attribute Values from MCattributeSet 89
Difference between setValue, addValue, and indexer 90
Assigning a NULL attribute value 90
Assigning a non-NULL attribute 90
Using an MCdataSource class to assign an attribute value 92
Retrieving Attribute Values 94
Using a Callback Class to Retrieve an Attribute's Value 95
Retrieving an Attribute Value's Properties 97
Listing an Attribute Set 97
Converting an Attribute Set into an XML string 98
Converting an XML string into an Attribute Set 99
8-bit Pixel Data 99
Encapsulated Pixel Data 100

Working with MCabstractMessage Derived Classes 101
Compression and Decompression 101
Merge DICOM Supplied Compressors and Decompressors 101
Validating Attribute Sets 105
The Overhead of Validation 109
Validating a Single Attribute 111
Streaming Attribute Sets 111
Message to XML conversion 114
XML to Message conversion 114

Message Exchange (Network Only) 115
Reading Network Messages 115
Using the MCdimseService 116
Using the sendRequestMessage method 116
Using the sendResponseMessage method 116

Using Attribute Containers 118
Using an Attribute Container in a Server Application 118
Using an Attribute Container in a Client Application 118
Declaring an MCattributeContainer Class 119
Writing the provideDataLength method 119
Writing the provideData method 120
Writing the receiveDataLength method 121
Writing the receiveData method 121
Writing the receiveMediaDataLength method 122
Registering Your MCattributeContainer 122
Releasing Your MCattributeContainer 123

Sequences of Items 123

DICOM Files 125
Constructing a new MCfile Instance 126
Construct an MCfile object with a pre-populated data set 126
Construct an MCfile object with an empty data set 126
Convert an MCdimseMessage object to an MCfile object 127
Accessing the service and command properties 127

Working with the contained file meta information 127
Accessing the File Preamble 128
Working with the contained data set 128
Resetting the MCfile object 128
File validation 129
The MCfile stream 129
Setting the file transfer syntax UID 129
Setting the file system file associated with the MCfile object 129
Listing the file's attributes 130
Using the MCmediaStorageService Class 130
Constructing an MCmediaStorageService object 131
Reading Files 131
Creating and Writing Files 133

The DICOMDIR file 134
Structure 134
Constructing a new MCdir Instance 135
The MCdirRecord class 135
Navigating the DICOMDIR 136
Adding and Deleting DICOMDIR Records 136

Memory Management 137
Assigning Pixel Data 137
Using Attribute Containers 138
Replacing Merge DICOM Toolkit's Memory Management Functions 139
Accessing Data When Needed 139
Saving Received Images Directly to Disk 139

DICOM Structured Reporting 141
Structured Report Structure and Modules 141
Content Item Types 143
Relationship Types between Content Items 145
Content Item Identifier 147
Observation Context 148
Structured Reporting Templates 149
Overview of the Merge DICOM Toolkit SR Classes 153
Encoding SR Documents 154
Reading SR Documents 157

Deploying Applications 158

Merge DICOM Required Files 158

Configuration Options 160

Appendix A: Frequently Asked Questions 163

Appendix B: Unique Identifiers (UIDs) 166

Summary of UID Composition 166

Obtaining a UID 167

Obtaining a UID From ANSI 167

Appendix C: Writing a DICOM Conformance Statement 168

Conformance Statement Sections 168
Application Data Flow 168
Sequencing of Real World Activities 169
AE Specifications 169
SOP Classes 170
Number of Associations 170
Asynchronous Nature 170
Implementation Identifying Information 170
SOP Specific Conformance 171
Transfer Syntax Selection Policies 171
Physical Network Interface 172
IPv4 and IPv6 Support 172
AE Title/Presentation Address Mapping 172
Configurable Parameters 172
PDU size 173
Standard Extended/Specialized/Private SOPs 173
Private Transfer Syntaxes 173

Appendix D: Configuration Parameters 174

Initialization File 174

Application Profile 176
Sections 176
Parameters 177

System Profile 188

Service Profile 209

Appendix E: XML structure 211

Base64 encoding of bulks and attributes with VR UN: 211

The default encoding of bulks and attributes with VR UN: 212

Overview

This User's Manual is targeted toward the developer of medical imaging applications using the Merge DICOM Toolkit™ to supply DICOM network or media functionality.

Merge DICOM Toolkit .NET supplies you with a powerful and simplified interface to DICOM. It lets you focus on the important details of your application and immediate needs of your end users, rather than the often complex and confusing details of the DICOM Standard.

The goal of this manual is to give you basic understanding of DICOM, and a clear understanding of the Merge DICOM Toolkit.

The DICOM Standard

The DICOM (Digital Imaging and Communications in Medicine) Standard was originally developed by a joint committee of the American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA) to "facilitate the open exchange of information between digital imaging computer systems in medical environments"[1].

Since its initial completion in 1993, the standard has taken hold. More and more products are advertising DICOM conformance, and more customers are requiring it. DICOM has also been incorporated as part of a developing European standard by CEN, as a Japanese standard by JIRA, and is increasingly becoming an International Standard.

DICOM Version 3.0 is composed of several hundreds of pages over sixteen separate parts. Each part of the standard focuses on a different aspect of the DICOM protocol:

Part 1: Introduction and Overview

Part 2: Conformance

Part 3: Information Object Definitions

Part 4: Service Class Specifications

Part 5: Data Structures and Encoding

Part 6: Data Dictionary

Part 7: Message Exchange

Part 8: Network Communication Support for Message Exchange

Part 9: Point-to-Point Communication Support for Message Exchange (retired)

Part 10: Common Media Storage Functions for Data Interchange

[1] NEMA Standards Publication No. PS 3.5-1993; DICOM Part 5 - Data Structures and Encoding, p.4.

Part 11: Media Storage Application Profiles

Part 12: Media Formats and Physical Media for Data Interchange

Part 13: Print Management Point-to-Point Communication Support (retired)

Part 14: Grayscale Standard Display Function

Part 15: Security Profiles

Part 16: DICOM Content Mapping Resource

Part 17: Explanatory Information

Part 18: Web Access to DICOM Persistent Objects (WADO)

A quick walk through DICOM

Part 1 of the standard gives an overview of the standard. Since this part was approved before most of the other parts were completed, it is already somewhat outdated and can be confusing.

Part 2 describes DICOM conformance and how to write a conformance statement. A conformance statement is important because it allows a network administrator to plan or coordinate a network of DICOM applications. For an application to claim DICOM conformance, it must have an accurate conformance statement.

Parts 3 and 4 define the types of services and information that can be exchanged using DICOM.

Parts 5 and 6 describe how commands and data shall be encoded so that decoding devices can interpret them.

Part 7 describes the structure of the DICOM commands, that along with related data, make up a DICOM message. This part also describes the association negotiation process, whereby two DICOM applications mutually agree upon the services they will perform over the network.

Part 8 describes how the DICOM messages are exchanged over the network using two prominent transport layer protocols; TCP/IP and OSI. (Note that IPv4 and IPv6 are supported by DICOM and by Merge DICOM Toolkit.). This is termed the DICOM Upper Layer Protocol (DICOM UL).

Part 9 is rarely of interest, as it describes how DICOM messages shall be exchanged using the 'old' 50-pin point-to-point connection originally specified in the predecessor to DICOM (ACR/NEMA Version 2). This part has been retired from the DICOM standard.

Part 10 describes the DICOM model for the storage of medical imaging information on removable media. It specifies the contents of a DICOM File Set, the format of a DICOM File and the policies associated with the maintenance of a DICOM Media Storage Directory (DICOMDIR) structure.

Part 11 specifies Media Storage Application Profiles that standardizes a number of choices related to a specific clinical need (modality or application). This includes the specification of a specific physical medium and media format (e.g., CD-ROM, 3.5" high-density floppy, …), as well as the types of information (objects) that can be stored within the DICOM File Set. Part 11 also includes useful templates to provide guidance in authoring media application conformance statements.

Part 12 details the characteristics of various physical medium and media formats that are referenced by the Media Storage Application Profiles of Part 11.

While parts 11 and 12 of DICOM are expected to evolve along with the introduction of new clinical procedures and the advancement of storage media and file system technology, Part 10 should remain quite stable since it specifies file formats independent of medical application or storage technology.

Part 13 details a point to point protocol for doing print management services. This part has been retired from the DICOM standard.

Part 14 specifies a standardized display function for display of grayscale images.

Part 15 specifies Security Profiles to which implementations may claim conformance. Profiles are defined for secure network transfers and secure media.

Part 16 specifies the DICOM Content Mapping Resource (DCMR) which defines the templates and context groups used elsewhere in the standard.

Part 17 consolidates informative information previously contained in other parts of the standard. It is composed of several annexes describing the use of the standard.

Part 18 specifies a web-based service for accessing and presenting DICOM persistent objects (e.g., images, medical imaging reports).

Figure 1 maps portions of the DICOM Standard dealing with networking to the ISO Open Systems Interconnection (OSI) basic reference model. The organization and terminology of the DICOM Standard corresponds closely with that used in the OSI Standard.

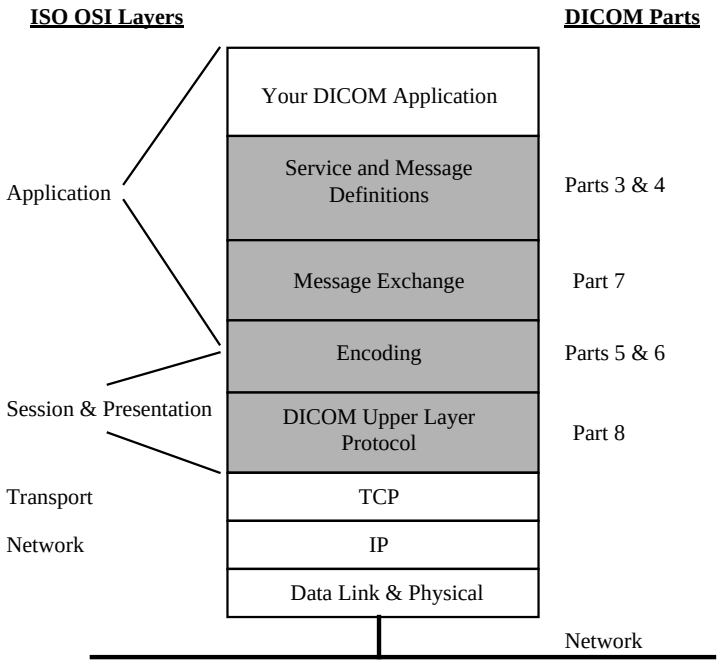

Figure 1: The DICOM Protocol Stack

Where to get the DICOM Standard

As a user of this toolkit, you should have access to the DICOM Standard. Merge DICOM Toolkit takes care of most of the details of DICOM for you. However, the standard is the final word. You will probably find Parts 2 – 6 most useful. The DICOM Standard can be ordered from:

NEMA
1300 N. 17th Street
Suite 1847
Rosslyn, VA 22209
USA
http://medical.nema.org

The DICOM Standard is typically published every year. Each version includes approved changes since the last publishing. The most recent version of the standard is available in PDF format and can be downloaded from NEMA's public ftp site at: ftp://medical.nema.org/medical/Dicom/2011

Special Note!

Please note that the DICOM Standard is evolving so rapidly, that additions to the Standard are published as 'supplements'. For example, the media extensions have been incorporated into the DICOM Standard as a supplement that contains addenda to various parts of the standard (e.g., PS3.3, PS3.4, …). If you find that this document references a part of the Standard and you cannot find what you are looking for in that part, you probably need to get the proper supplement from NEMA. Other additions to the Standard (e.g., new image objects or documents) will also be published as supplements. NEMA also makes all supplements to the standard freely available on their ftp server. You can reference these supplements at: ftp://medical.nema.org/medical/Dicom/Final/

The Merge DICOM Toolkit

Merge DICOM Toolkit provides a generalized implementation of DICOM in a .NET Assembly that you can use with your application. This .NET version of Merge DICOM makes use of the run-time library of the Merge DICOM C/C++ Toolkit. As such, it benefits from the power of that library while providing a complete .NET Assembly interface. You use methods of the Assembly to open connections with other DICOM devices on a network, and to build and exchange DICOM messages or DICOM files. The .NET Assembly is written in C# and all examples are supplied in C#, although it can be utilized from other .NET languages.

Figure 2 presents a pictorial representation of a DICOM Application Entity; Merge DICOM Toolkit implements for you everything in Parts 5, 6, 7, 8, and 10 of the DICOM Standard. It also makes it much easier for your application to implement according to Parts 3 and 4 by supplying many tools for the management of DICOM messages, and to Part 12 by supplying 'hooks' to your applications underlying file system.

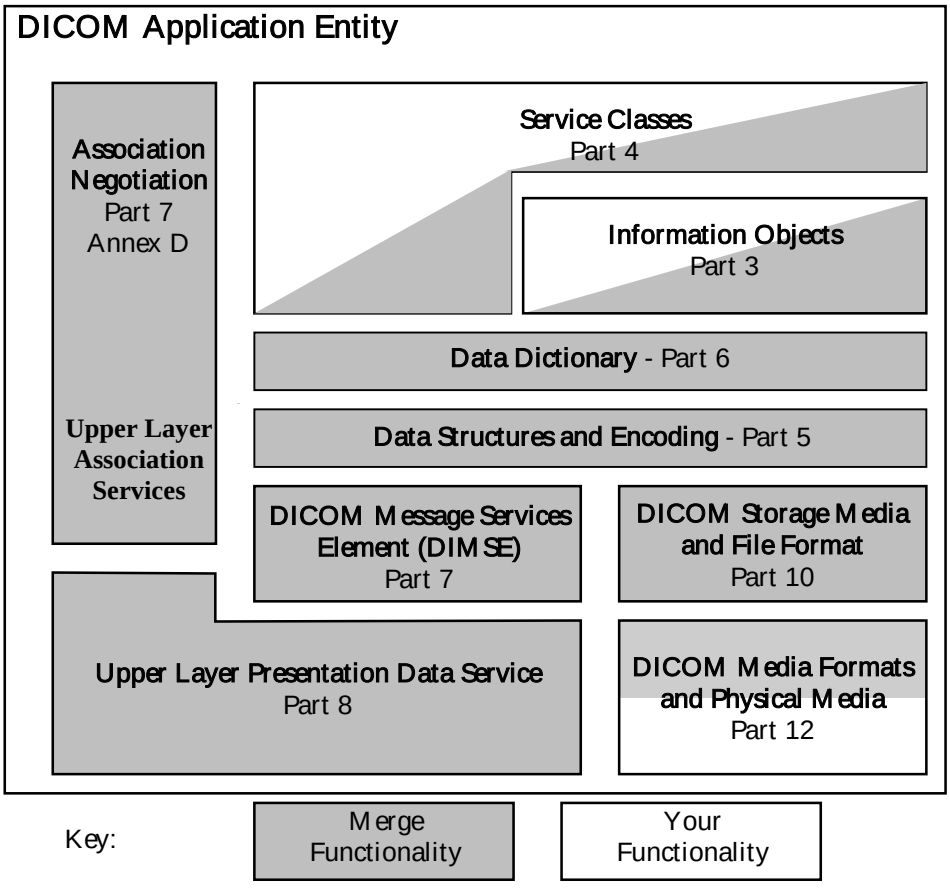
DICOM Application Entity
Association Negotiation Part 7 Annex D
Service Classes Part 4
Information Objects Part 3
Data Dictionary - Part 6
Upper Layer Association Services
Data Structures and Encoding - Part 5
DICOM Message Services Element (DIMSE) Part 7
DICOM Storage Media and File Format Part 10
Upper Layer Presentation Data Service Part 8
DICOM Media Formats and Physical Media Part 12
Key:
Merge Functionality
Your Functionality

Figure 2: The DICOM Application Layer

The DICOM Toolkit also supplies useful utility programs for testing a DICOM network connection, creating sample DICOM messages and writing them to a file, and for validating and listing the contents of DICOM messages.

Finally, sample application along with sample working source code give you valuable examples to work from when developing your own DICOM applications.

The DICOM Standard and the Merge Healthcare DICOM Toolkits allow applications to add private information to a DICOM message or file. For most application developers, this is more than sufficient. For applications that need to define their own non-standard private network or file services, an optional Merge DICOM Database Manual is available which describes the use of additional tools to extend the data dictionary.

Development Platform Requirements

The Merge DICOM Toolkit requires version 2.0 of the .NET Framework. The toolkit requires the C toolkit run-time library and currently requires 32-bit or 64-bit Microsoft Windows platforms (depending on what toolkit version you have) that supports the .NET framework.

Your development environment (or at a minimum your target environment) should run on a machine with a network interface over which you can run the TCP/IP protocol. The DICOM Toolkit library supplies you with the DICOM protocol that runs on top of TCP/IP.

If your application will write DICOM files to interchangeable media, you will need a device driver for the media storage device and a programming interface between your operating system and the file system on that device.

Assembly Structure

Understanding the organization and components of the Merge DICOM Assembly is important to developing an efficient and capable DICOM application (see *Figure 3*). Following is a description of the library's structure and the external components it uses at runtime to provide DICOM functionality.

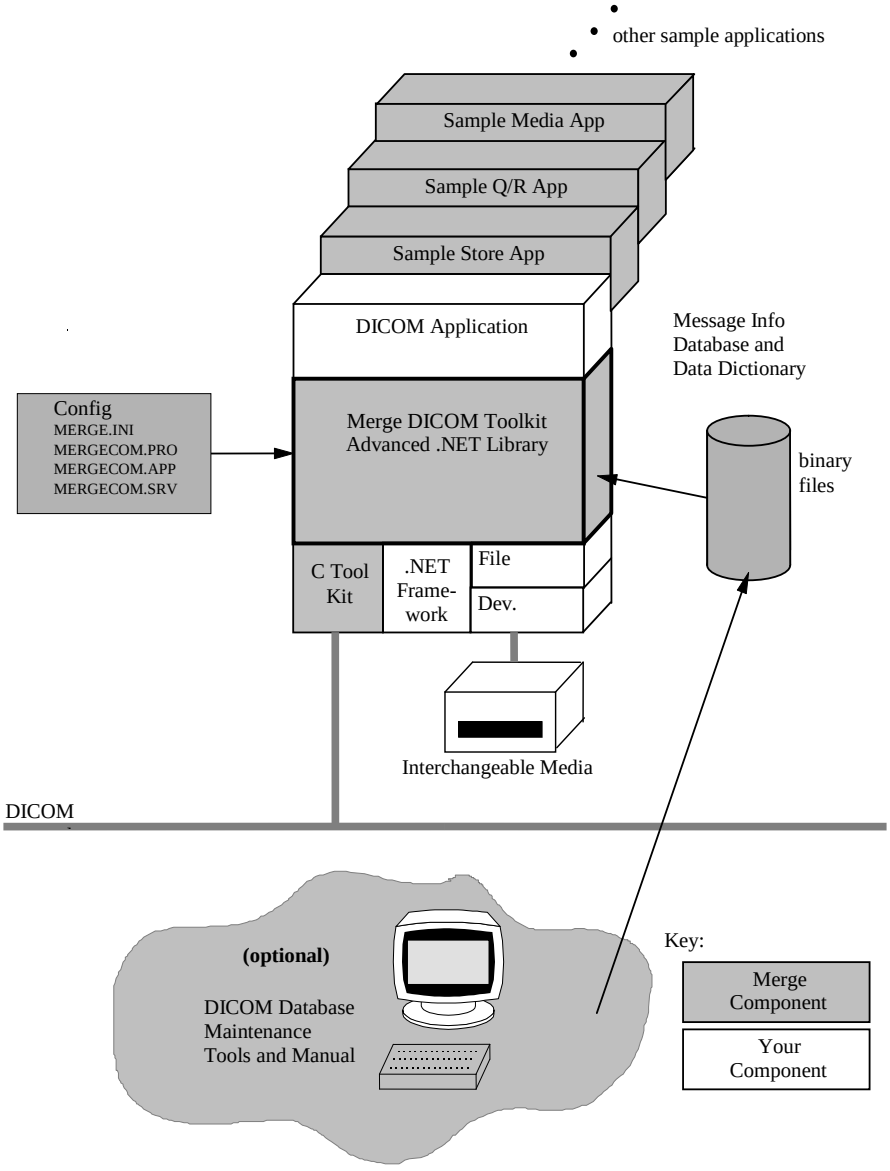

Figure 3: Merge DICOM Toolkit Library Organization

Merge DICOM C/C++ Toolkit Dynamic Library

The Merge DICOM C/C++ Toolkit Dynamic Library (usually named Mergecom.Native.dll for the Merge DICOM .NET/C# Toolkit) contains the core DICOM functionality required by the .NET toolkit. This library services many of the methods of the .NET DICOM Toolkit.

The Merge DICOM .NET/C# Toolkit Assembly and the Merge DICOM C/C++ Toolkit run-time library shipped with it, have been carefully designed to be re-entrant and have been validated to be thread-safe. The Merge DICOM Assembly automatically performs all DICOM network activity for each association instance in its own thread.

When a Merge DICOM Toolkit Application is first run, it reads in its configuration files; usually named merge.ini, mergecom.app, mergecom.pro, and mergecom.srv. Toolkit configuration is described later in this document. These configurable parameters are maintained in ASCII files for easy modification. When modifying your configuration files, your application must be re-run or the library reinitialized for those changes to take effect.

Binary Message Information and Data Dictionary Files

A great deal of the power of Merge DICOM Toolkit lies in its message handling and message validation capabilities. Message Objects are what is communicated between DICOM Application Entities. When your application creates a DICOM message object, the library accesses a binary message info file with information about that class of message. This info file describes to the library what attributes to expect as part of that message and each attribute's characteristics (Value Type, Conditions, and Enumerated or Defined Terms).

Another binary file containing the data dictionary is also accessed by the library. The data dictionary contains other characteristics of attributes (Name, Value Representation, and Value Multiplicity).

Performance Tuning

Merge DICOM Toolkit gives you added flexibility, by not requiring your application to make use of the message info file. Certain API calls allow you to open messages without accessing the info files. This means that the toolkit cannot validate your message against the DICOM standard, but this may not always be necessary once an application becomes stable. These options are discussed in detail in the Developing DICOM Applications section of this document.

Sample Applications

The sample applications can be a big help!

Included with the toolkit are sample C# applications and Visual Studio 2005 project files that compile the sample applications. Sample client and server applications are supplied for several DICOM services.

Merge DICOM Message Database Manual and Tools

Merge OEM has a DICOM Database Management System in which the DICOM standard is maintained. This database, along with a few additional tools, is used to generate the binary message info and dictionary files accessed by the DICOM Toolkit. As the DICOM standard is updated or extended, by simply maintaining this database, we can generate new binary files and keep the toolkit current. This also reduces the number of changes that must be made in the core DICOM Toolkit library over time.

A number of tools are included with Merge DICOM .NET/C# Toolkit for maintaining the data dictionary. A Message Database Manual is distributed with Merge DICOM and describes the use of these tools. This manual describes the use of the various tools and text files supplied with Merge DICOM Toolkit. Users can add definitions for private services and private attributes. Please reference this manual for further information on extending the Merge DICOM data dictionary.

Documentation Roadmap

The Merge DICOM Toolkit documentation is structured as pictured in *Figure 4*.

Read Me FIRST!

The User's Manual is the foundation for all other documentation because it explains the concepts of DICOM and the .NET/C# DICOM Toolkit. Before plunging into the Windows Help File, you should be comfortable with the material in the User Manual.

Assembly Reference

The Windows Help File serves as a reference manual for the .NET Assembly. This help file contains detailed information on the classes provided by the .NET/C# DICOM Toolkit.

The Release Notes contain a complete release history of the Merge DICOM Toolkit. It also contains a description of the software distribution and information on contacting Merge OEM for support.

Extension DB Manual

The DICOM Message Database Manual is an optional manual that describes the organization of the Merge DICOM Database and how to use it to extend standard services and define your own private services. Tools are supplied to integrate your changes and create a new binary runtime object database.

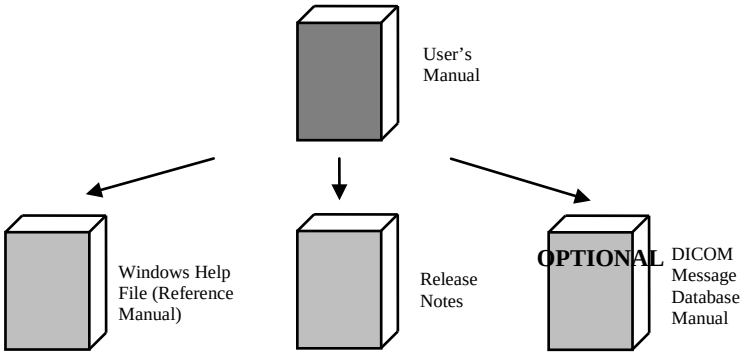

Figure 4: Merge DICOM Toolkit Documentation Roadmap

Conventions

This manual follows a few formatting conventions.

Terms that are being defined are presented in **boldface**.

Sample Margin Note

Margin notes (in the left margin) are used to highlight important points or sections of the document.

ClassName

When the descriptive text applies to one of the .NET classes provided by the DICOM Toolkit Library, the appropriate class name is listed in the left margin using Lucida Sans font.

Performance Tuning

Portions of the document that can relate directly to the performance of your application are marked with the special margin note **Performance Tuning*.***

Sample commands appear in **`bold courier`** font, while sample output, source code, and method calls appear in `standard courier` font.

Hexadecimal numbers are written with a trailing H. For example 16 decimal is equivalent to 10H hexadecimal.

Understanding DICOM

The many separate parts of the DICOM Standard can seem overwhelming, and most would agree that they are difficult to read. Part of what makes a successful standard is precision and detail. Our goal here is to explain the key concepts without delving too far into the detail, most of which is handled automatically for you by the DICOM Toolkit.

General Concepts

Some key concepts that must be understood to use the DICOM Toolkit wisely are common across both DICOM networking and interchangeable media applications. These concepts are discussed first.

Application Entities

MCapplication

The DICOM Standard refers extensively to **Application Entities (AE's)**. An application entity is simply a DICOM application. If your application interacts with other applications on a network or with interchangeable media using the DICOM protocol, it is an application entity.

Client/Server

DICOM also refers to **Service Class Users (SCU's)** and **Service Class Providers (SCP's)**. An application entity is an SCU when it requests DICOM services over a network and an SCP when it provides DICOM services over a network. We will more often refer to the SCU as a **Client** and the SCP as a **Server.** A single DICOM application entity can act as both a client and a server. This client/server model is a powerful and omnipresent one in the world of distributed network computing.

Services and Meta Services

MCdimseService (and it's subclasses)

MCfileService (and it's subclasses)

MCsopClass

DICOM is formed around the concepts of **Services** and **Service Classes**. The DICOM Standard specifies a set of services that can be performed over a network. Some of the services can also be stored to interchangeable media (these are italicized in *Table 1*). As new services are introduced, the standard will be further expanded. The standard also groups related services into a service class. *Table 1* lists the DICOM standard service classes and their component services. The DICOM Standard actually refers to services as Service Object Pairs (SOP's) and meta services as Meta-SOPs.

When a particular collection of services in a service class implies a higher level of service, this collection is combined by the standard into a **Meta Service**. Specifying that your application supports a specific meta service is a useful shorthand for explicitly listing out the collection of services that make up that meta service.

Table 1: DICOM Services Classes and their Component Services

Service Class	Services	Description
Verification **MCverificationService**	Verification	Verifies application level communication between DICOM application entities (AE's).
Storage **MCstorageService**	*Computed Radiography Image Storage* *CT Image Storage* *Enhanced CT Image Storage* *MR Image Storage* *Enhanced MR Image Storage* *MR Spectroscopy Storage* *Enhanced MR Color Image Storage* *Spatial Fiducials Storage* *Spatial Registration Storage* *Deformable Spatial Registration Storage* *Ultrasound Image Storage* *Ultrasound Multi-frame Image Storage* *Enhanced US Volume Storage* *Generic implant Template Storage* *Implant Assembly Template Storage* *Implant Template Group Storage* *Implantation Plan SR Document Storage* *Intraocular Lens Calculations Storage* *Macular Grid Thickness and Volume Report* *Nuclear Medicine Image Storage* *Positron Emission Tomography Image Storage* *Enhanced PET Image Storage* *Breast Tomosynthesis Image Storage* *Intravascular Optical Coherence Tomography Image Storage – For Presentation* *Intravascular Optical Coherence Tomography Image Storage – For Processing* *Digital X-Ray Image Storage – For Presentation* *Digital X-Ray Image Storage – For Processing* *Digital Intra-oral X-Ray Image Storage – For Presentation* *Digital Intra-oral X-Ray Image Storage – For Processing* *Digital Mammography Image Storage – For Presentation* *Digital Mammography Image Storage – For Processing* *Raw Data Storage*	Transfer of medical images and related standalone data between DICOM application entities, either over a network or using interchangeable media.

Service Class	Services	Description
	RT Beams Delivery Instruction Storage *RT Beams Treatment Record Storage* *RT Brachy Treatment Record Storage* *RT Dose Storage* *RT Plan Storage* *RT Image Storage* *RT Structure Set Storage* *RT Treatment Summary Record Storage* *RT Ion Beams Treatment Record Storage* *RT Ion Plan Storage* *VL Endoscopic Image Storage* *VL Microscopic Image Storage* *VL Photographic Image Storage* *VL Slide-Coordinates Microscopic Image Storage* *VL Whole Slide Microscopy Image Storage* *Video Endoscopic Image Storage* *Video Microscopic Image Storage* *Video Photographic Image Storage* *XA/XRF Grayscale Softcopy Presentation State Storage* *X-Ray Angiographic Image Storage* *X-Ray Radiofluoroscopic Storage* *X-Ray 3D Angiographic Image Storage* *X-Ray 3D Craniographic Image Storage* *Enhanced XA Image Storage* *Enhanced XRF Image Storage* *Secondary Capture Image Storage* *Autorefraction Measurements Storage* *Keratometry Measurements Storage* *Lensometry Measurements Storage* *Ophthalmic Axial Measurements Storage* *Ophthalmic Visual Field Static Perimetry Measurements Storage* *Subjective Refraction Measurements Storage* *Visual Acuity Measurements Storage* *Multi-frame Grayscale Byte Secondary Capture Image Storage* *Multi-frame Grayscale Word Secondary Capture Image storage* *Multi-frame Single Bit Secondary Capture Image Storage* *Multi-frame True Color Secondary Capture Image Storage*	

Service Class	Services	Description
	Segmentation Storage *Surface Segmentation Storage* *Basic Structured Display Storage* *Basic Text Structured Reporting* *Comprehensive Structured Reporting* *Enhanced Structured Reporting* *Key Object Selection* *Chest CAD SR* *Colon CAD SR* *Mammography CAD SR* *X-Ray Radiation Dose SR* *Encapsulated CDA Storage* *Encapsulated PDF Storage* *Procedure Log* *Blending Softcopy Presentation State Storage* *Color Softcopy Presentation State Storage* *Grayscale Softcopy Presentation State Storage* *Pseudo-Color Softcopy Presentation State Storage* *12-lead ECG Waveform Storage* *Ambulatory ECG Waveform Storage* *Arterial Pulse Waveform Storage* *Basic Voice Audio Waveform Storage* *Cardiac Electrophysiology Waveform Storage* *General Audio Waveform Storage* *General ECG Waveform Storage* *Hemodynamic Waveform Storage* *Ophthalmic 8 bit Photography Image Storage* *Ophthalmic 16 bit Photography Image Storage* *Ophthalmic Tomography Image Storage* *Spectacle Prescription Report Storage* *Stereometric Relationship Storage* *Real World Value Mapping Storage*	
MCstorageCommitment-Service Storage Commitment	Storage Commitment Push Storage Commitment Pull	Ensures that SOP Instances stored with the storage service class will not be deleted after reception but will be stored safely and can be retrieved again at a later point.

	Service Class	Services	Description
MCmediaStorageService	Media Storage	DICOM Basic Directory Storage and storage of various (italicized) services from the other Service Classes	Exists as a member of every DICOM File Set and contains general information about the file set and a hierarchical directory of the DICOM files contained in the file set.
MCqueryRetrieveService	Query/Retrieve	Patient Root Find Patient Root Move Patient Root Get Study Root Find Study Root Move Study Root Get Patient/Study Only Find Patient/Study Only Move Patient/Study Only Get	Management of images through a query and retrieval mechanism based on a small number of key attributes.
MCbasicWorklist-ManagementService	Basic Worklist Management	Modality Worklist Find	Supports the exchange of any type of worklist from one AE to another.
MCprintManagement-Service	Print Management	*Basic Film Session* *Basic Film Box* *Basic Grayscale Image Box* *Basic Color Image Box* Printer Printer Configuration Print Queue Management Pull Print Request Printer Referenced Image Box VOI LUT Box Presentation LUT Basic Annotation Box Basic Print Image Overlay Box SOP Class Print Job Image Overlay Retired **Basic Grayscale Print Mgmt. Meta** **Basic Color Print Mgmt. Meta** **Pull Stored Print Mgmt. Meta** **Ref. Grayscale Print Mgmt. Meta** **Ref. Color Print Mgmt. Meta**	Printing (or filming) of medical images and image related data on a hard copy medium. Also, storage of print related data to interchangeable media.
McstudyContent-NotificationService	Study Content Notification	Basic Study Content Notification	Allows one DICOM AE to notify another DICOM AE of the existence, contents, and source location of the images of a study.

Service Class	Services	Description
Patient Management	*Detached Patient Management* *Detached Visit Management* **Detached Patient Mgmt. Meta**	Creation and tracking of the subset of patient and patient visit information that is required to aid in the management of radiographic studies.
Study Management	*Detached Study Management* *Study Component Management* Modality Performed Procedure Step Modality Performed Procedure Step Notification Modality Performed Procedure Step Retrieve	Creation, scheduling, performance, and tracking of imaging studies.
Results Management	Detached Results Management Detached Interpretation Management **Detached Results Mgmt. Meta**	Creation and tracking of results and associated diagnostic interpretations.

MCpatientManagement-Service

MCstudyManagement-Service

MCresultsManagement-Service

Information Model

DICOM information model

The DICOM Standard includes the specification of a **DICOM Information Model**. A detailed entity-relationship diagram of this model is included in both parts 3 and 4 of the standard. This model specifies the relationship between the different types of objects (also called entities) managed in DICOM. For example, a Patient has one or more Studies, each of which are composed of one or more Series and zero or more Results, etc.

Objects vs. object instances

Most of DICOM's services perform actions on or with **object instances.**[2] An **object** can be thought of as a class of data (CT Image, Film Box, etc.) while an object instance is an actual occurrence of an object (a particular CT Image, a populated Film Box, etc.).

Normalized vs. composite

There are two types of objects (and hence, object instances) defined in DICOM. **Normalized objects** are objects consisting of a single entity in the DICOM information model (e.g., a Film Box). **Composite objects** are composed of several related entities (e.g., an MR Image). When possible, it is preferable to deal with normalized object instances over the network, because they contain less redundant data and can be more efficiently managed by an application.

Most services inherited from the ACR/NEMA Version 2.x Standard are **composite services** (operate on composite object instances) for reasons of backward compatibility. Newly introduced services, such as the HIS/RIS and Print Management Services, tend to be **normalized services** (operate on normalized object instances).

[2] object instances are referred to as SOP Instances or managed SOP's in the DICOM standard.

Networking

Certain aspects of DICOM only apply to networking when using the DICOM Toolkit. This includes networking commands and association negotiation.

Commands

DICOM defines a set of networking **commands**.[3] Each service uses a subset of these DICOM commands to perform the service over a network. These commands usually act on object instances. The C-commands operate on composite object instances, while the N-commands operate on normalized object instances.

The DICOM commands and brief descriptions of their actions are listed in *Table 2*.

Table 2: DICOM Commands

DICOM Commands	Description
C-STORE	Transfer an object instance to a remote AE.
C-GET	Retrieve object instance(s) from a remote AE whose attributes match a specified set of attributes.
C-MOVE	Move object instance(s) from a remote AE whose attributes match a specified set of attributes to yet another remote AE (or possibly your own AE - which would be another form of retrieval).
C-FIND	Match a set of attributes to the attributes of a set of object instances on a remote AE.
C-ECHO	Verify end-to-end communications with a remote AE.
N-EVENT-REPORT	Report an event to a remote AE.
N-GET	Retrieve attribute values from a remote AE.
N-SET	Request modification of attribute on a remote AE.
N-ACTION	Request an action by a remote AE.
N-CREATE	Request that a remote AE create a new object instance.
N-DELETE	Request that a remote AE delete an existing object instance.

[3] commands are referred to as DIMSE Services in the DICOM Standard.

Where your application takes over...

These DICOM commands can be thought of as primitives that every networking service is built from. In the context of a particular Service, these primitive actions translate to explicit real-world activities on the part of an Application Entity. Hence, DICOM places requirements on an application implementing a DICOM service. DICOM is careful to only express high-level operational requirements, and leaves the creative detail and look and feel of the application entity to the developer.

Request vs. response

For every command, there is both a **request** and a **response**. A command request indicates that a command should be performed and is usually sent to an SCP. A command response indicates whether a command completed or its state of completion and is usually returned to an SCU. Example request commands are C-STORE-RQ, N-GET-RQ, and N-SET-RQ. Example response commands are C-STORE-RSP, N-GET-RSP, and N-SET-RSP.

IMPORTANT!

It is important to note that this service definition level is where the Merge DICOM Toolkit Library leaves off, and your Application begins. While Merge DICOM supplies running sample applications source code for your platform, they are only supplied as an example. They clearly explain the requirements that implementing certain DICOM services places on your application and provide worthwhile but primitive examples of how to approach your application with the toolkit. While you will see that the toolkit saves you a great deal of 'DICOM work', it does not implement your end application for you.

Association Negotiation

MCasssciation

One of the areas where Merge DICOM Toolkit does a great deal of the 'DICOM work' for you is in opening an association (session) with another DICOM AE over the network. DICOM application entities need to agree on certain things before they operate with one another (open an association); these include:

- the services that can be performed between the two devices, which also impacts the commands and object instances that can be exchanged.
- the **transfer syntax** that shall be used in the network communication. The transfer syntax defines how the commands and object instances are encoded 'on the wire'.

The exchange of DICOM commands and object instances can only occur over an open association.

MCnegotiationInfo

DICOM defines an association negotiation protocol (see *Figure 5*) in which an **association requester** application proposes a connection with an **association acceptor** application. In the most common DICOM services, a client application entity (SCU) proposes an association with a server AE (SCP). However, some services define a mechanism where the client can be the SCP which opens an association with the SCU. This is used when an SCP sends asynchronous event reports to an SCU through the N-EVENT-REPORT command. This is done when DICOM role negotiation is used during standard association negotiation. For the sake of simplicity, the remainder of this manual refers to the client as the SCU and the server as the SCP.

MCproposedContext
MCproposedContextList
MCresultContext
MCsopClass

MCtransferSyntax
MCtransferSyntaxList

The association request proposal contains the set of services the client would like to perform and the transfer syntaxes it understands. The server then responds to the client with a subset of the services and transfer syntaxes proposed by the client. If this subset is empty, the server has rejected the association. If the subset is not empty, the server has accepted the association and the agreed upon services may be performed.

The client is responsible for releasing the association when it is finished performing its network operations. Either the client or the server can also abort the association in the case of some catastrophic failure (e.g., disk full, out of memory).

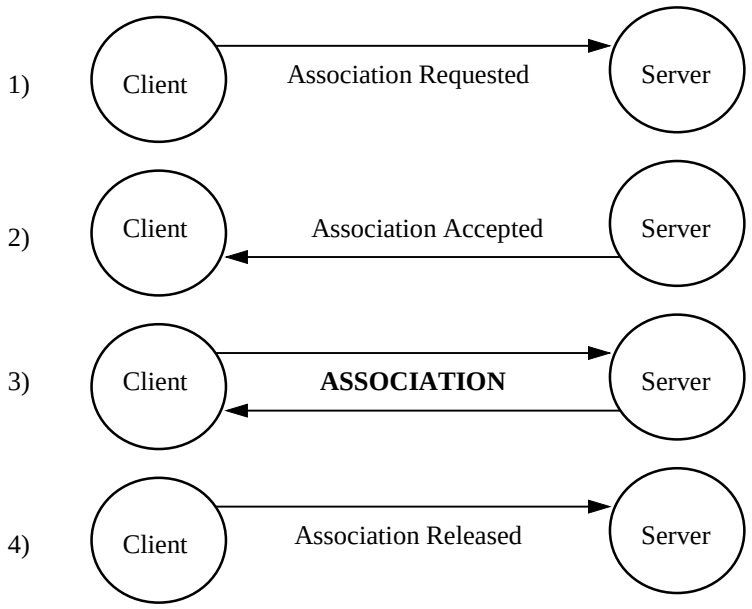

Figure 5: A Successful DICOM Association

Messages

MCdimseMessage
MCcommandSet
MCdataSet
MCitem

Once an association is established, services are performed by AE's through the exchange of DICOM **Messages**. A message is the combination of a DICOM command request or response and its associated object instance (see *Figure 6*). Messages containing command requests will be referred to as **request messages**, while messages containing command responses will be referred to as **response messages**.

MCfile
MCdir

When a DICOM service is stored to interchangeable media in a DICOM File, the structure of a DICOM File is a slightly specialized class of DICOM message. Media interchange is discussed in detail later; the only important thing to realize for now is that much of what is discussed relating to DICOM Messages also applies to DICOM Files.

MCsopClass

DICOM specifies the required message structure for each **service-command pair**. For example, the Patient Root Find - C-FIND-RQ service-command pair has a specific message structure. The command portion of a message is specified in Part 7 of the standard, while the object instance portion is specified in Parts 3 and 4.

MCattributeSet
MCattribute
MCdataElement
MCtag
MCdicom

The DICOM data dictionary defines many **data elements**. An **attribute** is a data element with a **value**. A message is constructed of attributes, with each attribute identified by a **tag**. An attribute is a unit of data (e.g., Patient's Name, Scheduled Discharge Date, ...). A tag is a 4 byte number identifying an attribute (e.g., 00100010H for Patient's Name, 0038001CH for Scheduled Discharge Date, ...).

MCtag.GroupNumber
MCtag.ElementNumber

A tag is usually written as an ordered pair of two byte numbers. The first two bytes are sometimes called a **group number**, with the last two bytes being called an **element number** (e.g., (0010, 0010), (0038, 001C), ...). This terminology is partly a remnant of the ACR-NEMA Standard where elements within a group were related in some manner. This can no longer be depended on in DICOM, but the ordered pair notation is still useful and often easier to read.

Also, the ordered pair notation is important when defining a Tag for a private attribute. We will see later that all private attributes must have an odd group number.

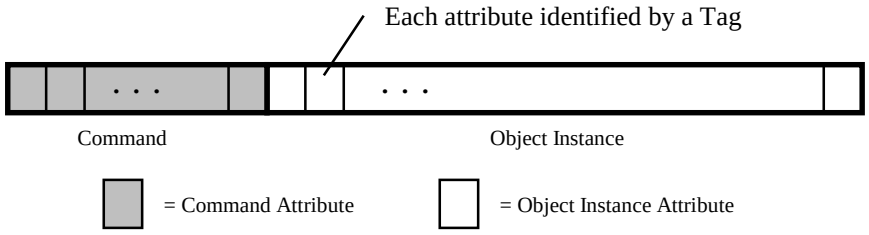

Figure 6: A DICOM Message

DICOM Data Dictionary

Value Representation
MCvr

Attributes have certain characteristics that apply to them no matter what message they are used in. These characteristics are specified in the DICOM Data Dictionary (Part 6 of DICOM) and are **Value Representation (VR)** and **Value Multiplicity (VM)**.

Value Representation can be thought of as the 'type specifier' for the values that can be assigned to an attribute. This includes the data type, as well as its format. The VR's defined by DICOM are listed in *Table 3*. You should refer to Part 5 of the standard for a detailed description of their legal values and formats.

Table 3: DICOM Value Representations (VR's)

VR	Name	VR	Name
AE	Application Entity	OF	Other Float String
AS	Age String	PN	Person Name
AT	Attribute Tag	SH	Short String
CS	Code String	SL	Signed Long
DA	Date	SQ	Sequence of Items
DS	Decimal String	SS	Signed Short
DT	Date Time	ST	Short Text
FL	Floating Point Single	TM	Time
FD	Floating Point Double	UI	Unique Identifier
IS	Integer String	UL	Unsigned Long
LO	Long String	UN	Unknown
LT	Long Text	US	Unsigned Short
OB	Other Byte String	UT	Unlimited Text
OW	Other Word String		

Value Multiplicity

A single attribute can have multiple values. Value Multiplicity defines the number of values an attribute can have. VM can be specified as 1, *k* , 1-*k*, or 1-n; where *k* is some integer value and n represents 'many'. For example, Part 6 specifies the VM of Scheduled Discharge Time (0038, 001D) as 1, while the VM of Referenced Overlay Plane Groups (2040, 0011) is 1-99.

Message Handling

MCattributeSet

Given the number of services and commands specified in *Table 1* and *Table 2*, it is clear that there are a great deal of messages to manage in DICOM. Remember, each service-command pair implies a different message. Fortunately, you will see later that Merge DICOM Toolkit saves the application developer a great deal of work in the message handling arena.

DICOM specifies the required contents of each message in Parts 3, 4, and 7 of the standard. For each attribute included in a message, additional characteristics of the attribute are defined that only apply within the context of a service. These characteristics are **Enumerated Values**, **Defined Terms,** and **Value Type**.

Enumerated values vs. defined terms

DICOM specifies that some attributes should have values from a specified set of values. If the attribute is an enumerated value, it shall have a value taken from the specified set of values. A good example of enumerated values are (M, F, O) for Patient's Sex (0010, 0040) in Storage services. If the attribute is a defined

term, it may take its value from the specified set, or the set may be extended with additional values. An example of defined terms are (CREATED, RECORDED, TRANSCRIBED, APPROVED) for Interpretation Status ID (4008, 0212) in Results Management services. If this set is extended by an application with another term, such as IN PROCESS, it should be documented in that application's conformance statement.

Value type

The most important characteristic of an attribute that is specified on a message by message basis, is the Value Type (VT). The VT of an attribute specifies whether or not that attribute needs to be included in a message and if it needs to have a value. Attributes can be required, optional, or only required under certain conditions (conditional attributes). Conditional attributes are always specified along with a condition. The value types defined by DICOM are listed in *Table 4*. Note that a null valued attribute has a value, that value being null (zero length).

Table 4: DICOM Value Types (VT's)

Value Type (VT)	Description
1	The attribute must have a value and be included in the message. The value cannot be null (empty).
1C	The attribute must have a value and be included in the message only under a specified condition. The value cannot be null. If that condition is not met, the attribute shall not be included in the message.
2	The attribute must have a value and be included in the message. If the value for the attribute is unknown and cannot be specified, its value shall be null.
2C	The attribute must have a value and be included in the message only under a specified condition. If the value for the attribute is unknown and cannot be specified, its value shall be null. If that condition is not met, the attribute shall not be included in the message
3	The attribute is optional. It may or may not be included in the message. If included, the attribute may or may not have a null value.

Private Attributes

MCfile
MCfileMetaInfo

The DICOM Standard allows application developers to add their own private attributes to a message as long as they are careful to follow certain rules. A **private attribute** is identified differently than are standard attributes. Its tag is composed of an odd group number, a private identification code string, and a single byte element number.

For example, ACME Imaging Inc. might define a private attribute to hold the name of the field engineer that last serviced their equipment. They could assign this attribute to private attribute tag (1455, 'ACME_IMG_INC', 00). This attribute has group number 1455, a private identification code string of 'ACME_IMG_INC', and a single byte element number of 00.

ACME could assign up 255 other private attributes to private group 1455 by using the other element numbers (01-FF). Part 5 of DICOM explains how these private tags are translated to standard group and element numbers and encoded into a message, while avoiding collisions. Merge DICOM Toolkit handles these details for you.

DICOM makes a couple of rules that must be followed when using private attributes:

- Private attributes shall not be used in place of required (Value Type 1, 1C, 2, or 2C) attributes.
- The possible value representations (VR's) used for private attributes shall be only those specified by the standard (see *Table 3*).

The way you use private attributes in your application can also greatly affect your conformance statement. DICOM conformance is discussed in greater detail later.

Media Interchange

MCmediaStorageService

The DICOM Standard specifies a DICOM file format for the interchange of medical information on removable media. This file format is a logical extension of the networking portion of the standard. When an object instance that was communicated over a network would also be of value when communicated via removable media, DICOM specifies the encapsulation of these object instances in a DICOM file.

DICOM Files

DICOM File Structure

MCfile
MCfileMetaInfo

A **DICOM File** is the encapsulation of a DICOM object instance, along with **File Meta Information**. File meta information is stored in the header of every DICOM file and includes important identifying information about the encapsulated object instance and its encoding within the file (see *Figure 7*).

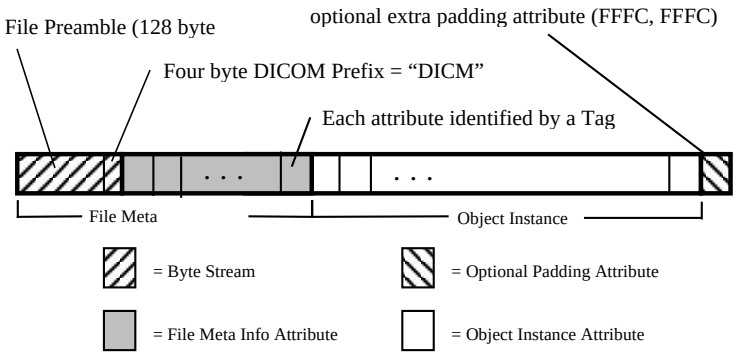
File Preamble (128 byte
optional extra padding attribute (FFFC, FFFC)
Four byte DICOM Prefix = "DICM"
Each attribute identified by a Tag
File Meta
Object Instance
= Byte Stream
= Optional Padding Attribute
= File Meta Info Attribute
= Object Instance Attribute

Figure 7: A DICOM File

The file meta information begins with a 128 byte buffer available for application profile or implementation specific use. **Application Profiles** standardize a number of choices related to a specific clinical need (modality or application) and are specified in Part 11 of the DICOM Standard. The next four bytes of the meta

information contain the DICOM prefix, which is always "DICM" in a DICOM file and can be used as an identifying characteristic for all DICOM files. The remainder of the file (preamble and object instance) is encoded using tagged attributes (as in a DICOM Message).

DICOM objects that can be written to media

The object instances that can be stored within the DICOM file are equivalent to a subset of the object instances that can be transmitted in network messages. The services that can be performed to interchangeable media are italicized in *Table 1*. The Media Storage Service Class (in Part 4 of the DICOM standard) specifies which service-command pairs can be performed to media. Remember it is the service-command pair that identifies the object instance portion of the message, and it is only the object instance portion of the message that is stored in a DICOM file. The command attributes associated with a network message are never stored in a DICOM File, only the data set.

The service command pairs whose corresponding object instances can be stored to media are summarized in *Table 5*. Note that the Media Storage Directory Service is not performed over a network and the single object specified in the Basic Directory Information Object Definition (Part 3) is used.

Table 5: Service-Command Pairs Specifying Objects that can be Stored in a DICOM File

Service	Command
Basic Structured Display Storage	C-STORE
Basic Text Structured Reporting	C-STORE
Breast Tomosynthesis Image Storage	C-STORE
Comprehensive Structured Reporting	C-STORE
Computed Radiography Image Storage	C-STORE
CT Image Storage	C-STORE
Chest CAD SR	C-STORE
Colon CAD SR	C-STORE
Deformable Spatial Registration Storage	C-STORE
Digital X-Ray Image Storage - For Presentation	C-STORE
Digital X-Ray Image Storage - For Processing	C-STORE
Digital Intra-oral X-Ray Image Storage - For Presentation	C-STORE
Digital Intra-oral X-Ray Image Storage - For Processing	C-STORE
Digital Mammography Image Storage - For Presentation	C-STORE
Digital Mammography Image Storage - For Processing	C-STORE
Encapsulated CDA Storage	C-STORE

Service	Command
Encapsulated PDF Storage	C-STORE
Enhanced CT Image Storage	C-STORE
Enhanced MR Image Storage	C-STORE
Enhanced MR Color Image Storage	C-STORE
Enhanced PET Image Storage	C-STORE
Enhanced US Volume Storage	C-STORE
Enhanced XA Image Storage	C-STORE
Enhanced XRF Image Storage	C-STORE
Enhanced Structured Reporting	C-STORE
Generic implant Template Storage	C-STORE
Grayscale Softcopy Presentation State Storage	C-STORE
Blending Softcopy Presentation State Storage	C-STORE
Color Softcopy Presentation State Storage	C-STORE
Pseudo-Color Softcopy Presentation State Storage	C-STORE
Implant Assembly Template Storage	C-STORE
Implant Template Group Storage	C-STORE
Implantation Plan SR Document Storage	C-STORE
Intraocular Lens Calculations Storage	C-STORE
Intravascular Optical Coherence Tomography Image Storage – For Presentation	C-STORE
Intravascular Optical Coherence Tomography Image Storage – For Processing	C-STORE
Key Object Selection	C-STORE
Macular Grid Thickness and Volume Report	C-STORE
Mammography CAD SR	C-STORE
MR Image Storage	C-STORE
MR Spectroscopy Storage	C-STORE
Multi-frame Grayscale Byte Secondary Capture Image Storage	C-STORE
Multi-frame Grayscale Word Secondary Capture Image Storage	C-STORE

Service	Command
Multi-frame Single Bit Secondary Capture Image Storage	C-STORE
Multi-frame True Color Secondary Capture Image Storage	C-STORE
Nuclear Medicine Image Storage	C-STORE
Positron Emission Tomography Image Storage	C-STORE
Procedure Log	C-STORE
Raw Data Storage	C-STORE
Real World Value Mapping Storage	C-STORE
RT Beams Delivery Instruction Storage	C-STORE
RT Beams Treatment Record Storage	C-STORE
RT Brachy Treatment Record Storage	C-STORE
RT Dose Storage	C-STORE
RT Ion Beams Treatment Record Storage	C-STORE
RT Ion Plan Storage	C-STORE
RT Plan Storage	C-STORE
RT Image Storage	C-STORE
RT Structure Set Storage	C-STORE
RT Treatment Summary Record Storage	C-STORE
Secondary Capture Image Storage	C-STORE
Segmentation Storage	C-STORE
Surface Segmentation Storage	C-STORE
Spatial Registration Storage	C-STORE
Spatial Fiducials Storage	C-STORE
Spectacle Prescription Report Storage	C-STORE
Standalone Overlay Storage	C-STORE
Standalone Curve Storage	C-STORE
Standalone Modality LUT Storage	C-STORE
Standalone VOI LUT Storage	C-STORE
Ultrasound Image Storage	C-STORE

Service	Command
Ultrasound Multi-frame Image Storage	C-STORE
Video Endoscopic Image Storage	C-STORE
Video Microscopic Image Storage	C-STORE
Video Photographic Image Storage	C-STORE
VL Endoscopic Image Storage	C-STORE
VL Microscopic Image Storage	C-STORE
VL Photographic Image Storage	C-STORE
VL Slide-Coordinates Microscopic Image Storage	C-STORE
VL Whole Slide Microscopy Image Storage	C-STORE
XA/XRF Grayscale Softcopy Presentation State Storage	C-STORE
X-Ray Angiographic Image Storage	C-STORE
X-Ray Radiofluoroscopic Image Storage	C-STORE
X-Ray Radiation Dose SR Storage	C-STORE
X-Ray 3D Angiographic Image Storage	C-STORE
X-Ray 3D Craniofacial Image Storage	C-STORE
12-lead ECG Waveform Storage	C-STORE
Ambulatory ECG Waveform Storage	C-STORE
Arterial Pulse Waveform Storage	C-STORE
Basic Voice Audio Waveform Storage	C-STORE
Cardiac Electrophysiology Waveform Storage	C-STORE
General Audio Waveform Storage	C-STORE
General ECG Waveform Storage	C-STORE
Hemodynamic Waveform Storage	C-STORE
Respiratory Waveform Storage	C-STORE
Hanging Protocol Storage	C-STORE
Ophthalmic 8 bit Photography Image Storage	C-STORE
Ophthalmic 16 bit Photography Image Storage	C-STORE
Ophthalmic Tomography Image Storage	C-STORE

Service	Command
Stereometric Relationship Storage	C-STORE
Detached Patient Management	N-GET
Detached Visit Management	N-GET
Detached Study Management	N-GET
Detached Study Component Management	N-GET
Detached Results Management	N-GET
Detached Interpretation Management	N-GET
Basic Film Session	N-CREATE
Basic Film Box	N-CREATE
Basic Grayscale Image Box	N-SET
Basic Color Image Box	N-SET
Autorefraction Measurements Storage	C-STORE
Keratometry Measurements Storage	C-STORE
Lensometry Measurements Storage	C-STORE
Subjective Refraction Measurements Storage	C-STORE
Ophthalmic Axial Measurements Storage	C-STORE
Ophthalmic Visual Field Static Perimetry Measurements Storage	C-STORE
Visual Acuity Measurements Storage	C-STORE
Media Storage Directory Storage	C-STORE*

* Merge DICOM Toolkit defines a C-STORE command for the DICOMDIR service even though it does not formally exist In the DICOM Standard.

Finally, the DICOM file can be padded at the end with the Data Set Trailing Padding attribute (FFFC, FFFC) whose value is specified by the standard to have no significance.

File Sets

MCfile

DICOM Files must be stored on removable media in a **DICOM File Set.** A DICOM file set is defined as a collection of DICOM files sharing a common naming space within which file ID's are unique (e.g., a file system partition). A **DICOM File Set ID** is a string of up to 16 characters that provides a name for the file set.

A **File ID** is a name given to a DICOM file that is mapped to each media format specification (in Part 12 of DICOM). A file ID consists of an ordered sequence of one to eight components, where each component is a string of one to eight characters. One can certainly imagine mapping such a file ID to a hierarchical file system, and this is done for several media formats in Part 12. It is important to note that DICOM states that no semantic relationship between DICOM files shall be conveyed by the contents or structure of file ID's (e.g., the hierarchy). This helps insure that DICOM files can be stored in a media format and file system independent manner.

Naming DICOM File Sets and File ID's

The allowed characters in both a file ID's and file set ID's are a subset of the ASCII character set consisting of the uppercase characters (A-Z), the numerals (0-9), and the underscore (_).

The DICOMDIR

MCdir

The **DICOM Directory File** or DICOMDIR is a special type of a DICOM File. A single DICOMDIR must exist within each DICOM file set, and is always given the file ID "DICOMDIR". It is the DICOMDIR file that contains identifying information about the entire file set, and usually (dependent on the Application Profile) a directory of the file set's contents.

Figure 8 shows a graphical representation of a DICOMDIR file and its central role within a DICOM File Set.

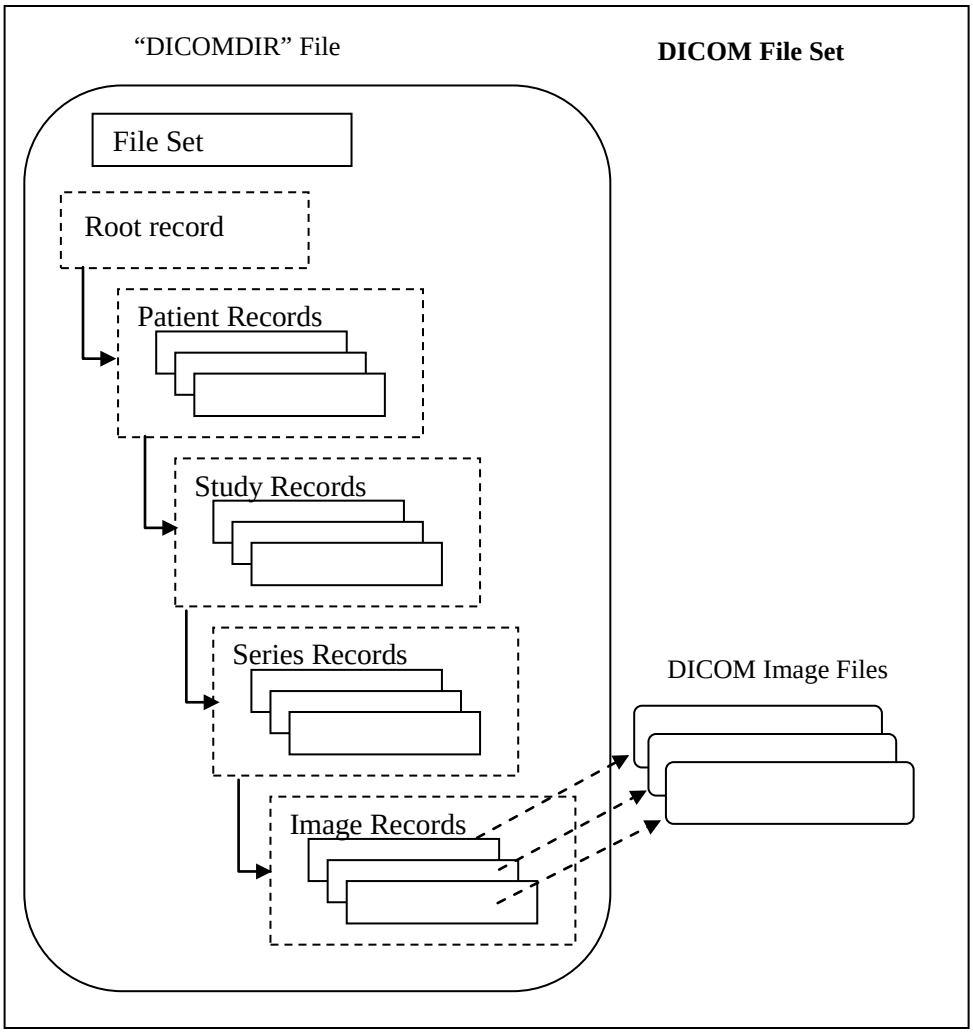

Figure 8: A DICOM Directory File (DICOMDIR) within a DICOM File Set

The DICOMDIR hierarchy

If the DICOMDIR file contains directory information, it is composed of a hierarchy of directory records, with the top-most directory record being the root directory record. A **Directory Record** identifies a DICOM File by summarizing key attributes and their values in the file and specifying the file ID of the corresponding file. The file ID can then be used, in the context of the native file system, to access the corresponding DICOM file. Each directory record can in turn point down the hierarchy to one or more related directory records.

Part 3 of the DICOM Standard specifies the allowed relationships between directory records in the section defining the Basic Directory IOD. We reproduce this table here (see *Table 6*) for pedagogical reasons; but, you should refer to the DICOM Standard for the most up-to-date and accurate specification.

Table 6: Allowed Directory Entity

Directory Record Type	Record Types which may be included in the next lower-level Directory Entity
(Root Directory Entity) *	PATIENT, TOPIC, PRINT QUEUE, HANGING PROTOCOL, PRIVATE
PATIENT	STUDY, PRIVATE
STUDY	SERIES, VISIT, RESULTS, STUDY COMPONENT, FILM SESSION, PRIVATE
SERIES	IMAGE, STORED PRINT, RT DOSE, RT STRUCTURE SET, RT PLAN, RT TREAT RECORD, OVERLAY, MODALITY LUT, VOI LUT, CURVE, SR DOCUMENT, PRESENTATION, KEY OBJECT DOC, SPECTROSCOPY, RAW DATA, WAVEFORM, REGISTRATION, FIDUCIAL, VALUE MAP, ENCAP DOC, PRIVATE
HANGING PROTOCOL	PRIVATE
IMAGE	PRIVATE
STORED PRINT	PRIVATE
RT DOSE	PRIVATE
RT STRUCTURE SET	PRIVATE
RT PLAN	PRIVATE
RT TREAT RECORD	PRIVATE
OVERLAY	PRIVATE
MODALITY LUT	PRIVATE
VOI LUT	PRIVATE
CURVE	PRIVATE
SR DOCUMENT	PRIVATE
PRESENTATION	PRIVATE
KEY OBJECT DOC	PRIVATE
SPECTROSCOPY	PRIVATE
RAW DATA	PRIVATE
WAVEFORM	PRIVATE
REGISTRATION	PRIVATE
FIDUCIAL	PRIVATE
VALUE MAP	PRIVATE

Directory Record Type	Record Types which may be included in the next lower-level Directory Entity
ENCAP DOC	PRIVATE
TOPIC	STUDY, SERIES, IMAGE, OVERLAY, MODALITY LUT, VOI LUT, CURVE, FILM SESSION, PRIVATE
VISIT	PRIVATE
RESULTS	INTERPRETATION, PRIVATE
INTERPRETATION	PRIVATE
STUDY COMPONENT	PRIVATE
PRINT QUEUE	FILM SESSION, PRIVATE
FILM SESSION	FILM BOX, PRIVATE
FILM BOX	BASIC IMAGE BOX, PRIVATE
BASIC IMAGE BOX	PRIVATE
PRIVATE	PRIVATE
MRDR	(Not applicable)

*The first row of this table specifies the directory records that can be contained within the Root Directory Entity.

File Management Roles and Services

File Management Services

Part 10 of the DICOM Standard specifies a set of file management roles and services. There are five **DICOM File Services**, that describe the entire set of DICOM file operation primitives:

Table 7: DICOM File Services

DICOM File Services	Description
M-WRITE	Create new files in a file set and assign them a file ID.
M-READ	Read existing files based on their file ID.
M-DELETE	Delete existing files based on their file ID.
M-INQUIRE FILE-SET	Inquire free space available for creating new files within a file set.
M-INQUIRE FILE	Inquire date and time of file creation (or last update if applicable) for any file within a file set.

The Merge DICOM Toolkit supplies the MCmediaStorageService that performs the first two (underlined) file services. That class also implements enhanced read and write functionality for the creation and maintenance of DICOMDIR files and its hierarchy of directory entities and directory records. The remaining three file services are best implemented by the application entity through file system calls because they are file system dependent operations.

File Management Roles

DICOM AE's that that perform file interchange functionality are in turn classified into three roles:

File Set Creator (FSC) — uses M-WRITE operations to create a DICOMDIR file and one or more DICOM files.

File Set Reader (FSR) — uses M-READ operations to access one or more files in a DICOM file set. An FSR shall not modify any files of the file set (including the DICOMDIR file).

File Set Updater (FSU) — performs M-READ, M-WRITE, and M-DELETE operations. It reads, but shall not modify the content of any DICOM files other than the DICOMDIR file. It may create additional files by means of an M-WRITE or delete existing files by means of an M-DELETE.

The concept of these roles is used within the DICOM conformance statement of an application entity that supports media interchange to more precisely express the capabilities of the implementation. Conforming applications shall support one of the capability sets specified in *Table 8*. DICOM conformance is described in greater detail in the next section.

Table 8: Media Application Operations and Roles

Media Roles	M-WRITE	M-READ	M-DELETE	M-INQUIRE FILE-SET	M-INQUIRE FILE
FSC	Mandatory	not required	not required	Mandatory	Mandatory
FSR	not required	Mandatory	not required	not required	Mandatory
FSC+FSR	Mandatory	Mandatory	not required	Mandatory	Mandatory
FSU	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory
FSU+FSC	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory
FSU+FSR	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory
FSU+FSC+FSR	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory

Conformance

Part 2 of DICOM discusses conformance and is important to any AE developer. For an application to be DICOM conformant it must:

- meet the minimum general conformance requirements specified in Part 2 and service specific conformance requirements specified in Part 4 (Network Services), and/or Parts 10 and 11 (Media Services).
- have a published DICOM conformance statement detailing the above conformance and any optional extensions.

Conformance also applies to aspects of the communications protocol that are managed by the DICOM Toolkit. Most parameters are configurable by your application. The conformance statement for the Merge DICOM Toolkit in Appendix C: DICOM Conformance Statement lists all these protocol parameters and how they can be configured.

Part 2 also deals with private extensions to the DICOM Standard by defining **Standard Extended Services**. Standard Extended Services give your application a little more flexibility, by allowing you to add private attributes as long as they are of value type 3 (optional) and are documented in the conformance statement.

DICOM also allows you to define your own **Specialized** and **Private Services**. These should be avoided by most applications since they are non-standard, add complexity to your application, and limit interoperability.

If you are significantly extending services or creating your own private services, you may need the Merge DICOM Toolkit Extended Toolkit to assist in defining these services so that they can be supported by the toolkit.

Using the Merge DICOM Toolkit

You can use the Merge DICOM Toolkit 'out of the box' by using its supplied utility programs and sample applications. In this section we discuss how to configure the toolkit and to use the utility programs. Later, we discuss how to develop your own DICOM applications using the Merge DICOM .NET™ Assembly.

Configuration

Merge DICOM is highly configurable, and understanding its configuration files is critical to using the library effectively. Four configuration files are used by Merge DICOM: an initialization file, an application profile, a system profile, and a service profile

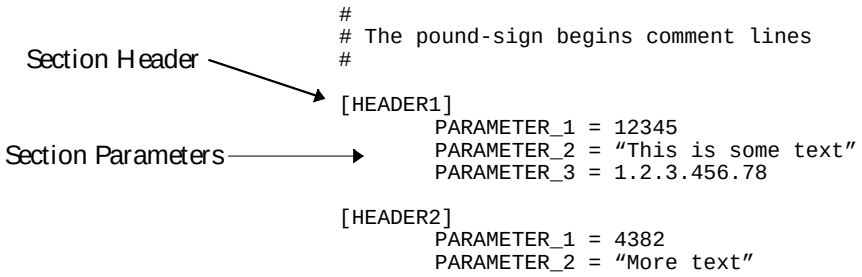

Figure 9: Format of a configuration file

Each of the four toolkit initialization files follow the same format. The format of the initialization files is the same format that is used by others in the industry. Configuration files are broken down into sections for easier organization and grouping of parameters. Each section has a section heading enclosed in square brackets. Next, parameters are defined by putting the parameter name to the left of an equal sign and its initial value to the right. Zero of more spaces may precede and follow the equal sign. *Figure 9* illustrates the format of an "ini" file.

Notice that parameter names are relative to their header sections. For example, `PARAMETER_1` and `PARAMETER_2` are defined twice in the above example "ini" file. But, since each is defined in a different section, they are considered different entities.

Each of the four configuration files are discussed separately below. Only the key configurable parameters are summarized here. For detailed descriptions of all configuration files and their parameters, see Appendix D: Configuration Parameters.

Initialization File

The Merge DICOM Initialization File (usually called `merge.ini`) provides the DICOM Toolkit with its top-level configuration. It specifies the location of the other three configuration files, along with message and error logging characteristics.

All Merge DICOM applications require access to the Initialization File. Your .NET programs access the Initialization File differently than do C applications, such as the utility programs distributed with the Merge DICOM Toolkit.

MERGE_INI environmental variable needed only by utility programs

The C utility programs access the merge.ini file by accessing the `MERGE_INI` environment variable. You must set the `MERGE_INI` environmental variable to point to the Initialization File. This variable can be set within a command shell; for example:

In DOS command shell:

```
set MERGE_INI=\mc3adv\merge.ini
```

See the Platform notes for your platform if none of these methods apply.

Use MC.mcInitialization to provide Initialization File location

Merge DICOM applications written in .NET do not use the `MERGE_INI` environment variable. Instead, they determine the location of the Initialization File in any way that is appropriate and then pass the location to the Assembly, using the static mcInitialization method of the MC class.

The initialization file contains one `[MergeCOM3]` section that points to the location of the other three Merge DICOM initialization files, specifies characteristics of the message/error log kept by the DICOM Toolkit library, turns particular types of logging on and off, and specifies where the messages are logged (file, screen, both, or neither). In most cases the INFO, WARNING, and ERROR messages will be sufficient. The `T`*n*`_MESSAGE` settings (where *n* is an integer between 1 and 9) turns on lower-level protocol tracing capabilities. These capabilities can prove useful when running into difficulties communicating with other implementations of DICOM over a network and can be used by Merge OEM service engineers in diagnosing lower-level network problems.

Application Profile

The Merge DICOM Application Profile (usually called `mergecom.app`) specifies the characteristics of your own application entity and the AE's your application will connect with over a network. The name and location of this file is specified in the `[MergeCOM3]` section of the Merge DICOM initialization file.

DICOM AE Title

When your application acts as a client (SCU), you must specify in the Application Profile the network address of the server (SCP) Application Entities you wish to connect (open an association) with. Your client refers to the application entity by a **DICOM Application Entity Title** and this is the same way it is referred to in the application profile. The AE title consists of a string of characters containing no spaces and having a length of 16 characters or less. A section of the profile exists for each Server AE you wish to connect with.

For example, if your application is an image source and also performs query and retrieval of images from two separate DICOM AE's, it might contain sections like the following:

```
[Acme_Store_SCP]
   PORT_NUMBER     = 104
   HOST_NAME       = acme_sun1
   SERVICE_LIST    = Storage_Service_List
[Acme_QR_SCP]
   PORT_NUMBER     = 104
   HOST_NAME       = acme_hp2
   SERVICE_LIST    = Query_Service_List
```

`Acme_Store_SCP` and `Acme_QR_SCP` are the AE titles for the applications you wish to connect with. The storage server runs on a Sun computer having the host name `acme_sun1`, while the query/retrieve server runs on an HP workstation with the host name `acme_hp2`. Both servers listen on port 104 (the standard DICOM listen port). The host name and port combined, make up the TCP/IP network address for a listening server application. See *Figure 10*.

Besides entering a hostname for the HOST_NAME parameter, it is also possible to simply enter an IP address. Both IPv4 addresses and IPv6 addresses are allowed in this field.

Service List

The `SERVICE_LIST` is set to the name of another section in the application profile that lists the DICOM services that will be negotiated with that application entity. For example, in this case these sections might look like:

```
[Storage_Service_List]
   SERVICES_SUPPORTED = 11 # Services in list
   SERVICE_1               = STANDARD_MR
   SERVICE_2               = STANDARD_CR
   SERVICE_3               = STANDARD_CT
   SERVICE_4               = STANDARD_CURVE
   SERVICE_5               = STANDARD_MODALITY_LUT
   SERVICE_6               = STANDARD_OVERLAY
   SERVICE_7               = STANDARD_SEC_CAPTURE
   SERVICE_8               = STANDARD_US
   SERVICE_9               = STANDARD_US_MF
   SERVICE_10              = STANDARD_VOI_LUT
   SERVICE_11              = STANDARD_NM
[Query_Service_List]
   SERVICES_SUPPORTED = 2  # Services in list
   SERVICE_1               = STUDY_ROOT_FIND
   SERVICE_2               = STUDY_ROOT_MOVE
```

`[Storage_Service_List]` lists the storage services that will be requested, while `[Query_Service_List]` lists the type of query/retrieve that will be requested. These service names are the strings used in Merge DICOM Toolkit to identify standard DICOM services. Any services listed must be defined in the Service Profile, discussed below.

Dynamic Service Lists
MCproposedContextList

You may also dynamically create service lists at run time, using the methods of the MCproposedContext and MCproposedContextList classes, as well as the constructors for the MCproposedContextList class. This will be discussed in more detail in the DEVELOPING DICOM APPLICATIONS section below.

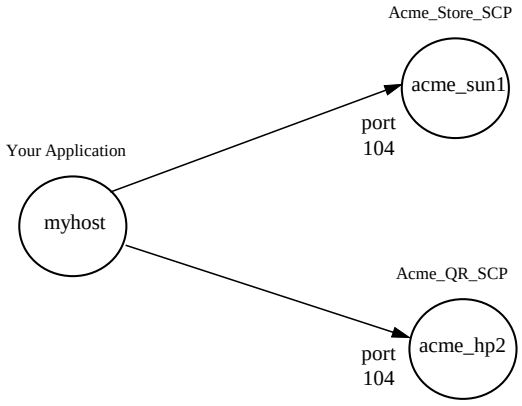

Figure 10: An example configuration of DICOM applications

Don't forget!

A service list also needs to be defined for each of your own server AE's. Even though you do not need a section for your server AE Title (since it is running on your local machine), you do need to specify a service list that your application supports as an SCP. If your application also acts as a storage server, for example, it could use [`Storage_Service_List`]. You also need to specify a listen port for your server AE in the System Profile, which is discussed below.

Transfer Syntax List

For DICOM Toolkit users, Merge DICOM allows for the defining of the transfer syntaxes supported for each service in a service list. This functionality is implemented through the use of transfer syntax lists. The basic service lists discussed above can be modified to include these transfer syntax lists. The following is an example service list that has transfer syntaxes specified for each service:

```
[Storage_Service_List]
   SERVICES_SUPPORTED   = 3  # Number of Services
   SERVICE_1            = STANDARD_MR
   SYNTAX_LIST_1        = MR_Syntax_List
   SERVICE_2            = STANDARD_US
   SYNTAX_LIST_2        = US_Syntax_List
   SERVICE_3            = STANDARD_CT
   SYNTAX_LIST_3        = CT_Syntax_List

[MR_Syntax_List]
   SYNTAXES_SUPPORTED   = 4  # Number of Syntaxes
   SYNTAX_1             = JPEG_BASELINE
   SYNTAX_2             = EXPLICIT_BIG_ENDIAN
   SYNTAX_3             = EXPLICIT_LITTLE_ENDIAN
   SYNTAX_4             = IMPLICIT_LITTLE_ENDIAN

[US_Syntax_List]
   SYNTAXES_SUPPORTED   = 2  # Number of Syntaxes
   SYNTAX_1             = RLE
   SYNTAX_2             = IMPLICIT_LITTLE_ENDIAN

[CT_Syntax_List]
   SYNTAXES_SUPPORTED   = 2  # Number of Syntaxes
   SYNTAX_1             = EXPLICIT_LITTLE_ENDIAN
   SYNTAX_2             = IMPLICIT_LITTLE_ENDIAN
```

[Storage_Service_List] lists some standard storage service class services used by Merge DICOM Toolkit. The SYNTAX_LIST_N parameter has been added to this example to specify a transfer syntax list for each service. This optional parameter is set to the name of another section in the application profile which lists a group of DICOM transfer syntaxes to be negotiated. When this parameter is not set, the default non-compressed transfers syntaxes (implicit VR little endian, explicit VR little endian, and explicit VR big endian) are negotiated.

The [MR_Syntax_List], [US_Syntax_List], and [CT_Syntax_List] sections each define a separate transfer syntax list for the MR, US, and CT services respectively. Merge DICOM Toolkit currently supports all transfer syntaxes specified in the DICOM standard. The names used for these transfer syntaxes are defined in *Table 33*, in Appendix D: Configuration Parameters.

MCtransferSyntax MCtransferSyntaxList

As mentioned earlier, Merge DICOM supports the dynamic creation of service lists at runtime. The Assembly also provides the MCtransferSyntaxList.getObject factory method to construct an MCtransferSyntaxList object from information in the configuration files. This will be discussed in more detail in the DEVELOPING DICOM APPLICATIONS section below.

Transfer syntax priority during association negotiation

For server (SCP) applications, the order in which transfer syntaxes are specified in a transfer syntax list dictates the priority Merge DICOM places on them during association negotiation. For example, in the [US_Syntax_List] specified above, if a client (SCU) proposed the Ultrasound storage service with the RLE compressed transfer syntax and the implicit VR little endian transfer syntax, Merge DICOM would select the RLE transfer syntax because it was listed first in the transfer syntax list.

When a transfer syntax list is not specified in a service list the priority Merge DICOM Toolkit places on transfer syntaxes during association negotiation is dependent on the hardware platform. On little endian machines (Intel based systems) the priority order is: Explicit VR Little Endian, Implicit VR Little Endian, and Explicit VR Big Endian.

Merge DICOM also supports DICOM role negotiation through its service lists. Whereas in previous examples, the same service list could be used for both client (SCU) and server (SCP), these service lists are specific to the role to be negotiated for each service.

```
[SCU_Service_List]
   SERVICES_SUPPORTED   = 1  # Number of Services
   SERVICE_1            = STORAGE_COMMITMENT_PUSH
   ROLE_1               = SCU

[SCP_Service_List]
   SERVICES_SUPPORTED   = 1  # Number of Services
   SERVICE_1            = STORAGE_COMMITMENT_PUSH
   ROLE_1               = SCP
```

In this case, the [SCU_Service_List] supports the Storage Commitment Push SOP class as an SCU and the [SCP_Service_List] supports the Storage Commitment Push SOP class as an SCP. Merge DICOM will negotiate the association based on the settings for these roles.

The role for a service can be defined as SCU, SCP, BOTH, or be undefined. *Table 9* contains a complete listing of configurable roles for both requestors and acceptors along with the resultant negotiated roles. Note that in some cases a service will be rejected because the roles being negotiated do not match.

Table 9: Negotiated Roles

Requestor's Configured Role	Acceptor's Configured Role	Requestor's Negotiated Role	Acceptor's Negotiated Role
SCU	SCP	SCU	SCP
	SCU	Rejected	Rejected
	BOTH	SCU	SCP
	NOT DEFINED	SCU	SCP
SCP	SCP	Rejected	Rejected
	SCU	SCP	SCU
	BOTH	SCP	SCU
	NOT DEFINED	Rejected	Rejected
BOTH	SCP	SCU	SCP
	SCU	SCP	SCU
	BOTH	BOTH	BOTH
	NOT DEFINED	SCP	SCP
NOT DEFINED	SCP	SCU	SCP
	SCU	Rejected	Rejected
	BOTH	SCU	SCP
	NOT DEFINED	SCU	SCP

For detailed information about the content of the Application Profile, see Application Profile in Appendix D: Configuration Parameters.

Configuring Asynchronous Communications Support

DICOM Asynchronous Communication

Merge DICOM also optionally supports DICOM Asynchronous Operations Window Negotiation through service lists. The same service list can be used in this case for both the client (SCU) and server (SCP). The following is an example service list that configures DICOM asynchronous communication negotiation:

```
[SCU_Or_SCP_Service_List]
   SERVICES_SUPPORTED         = 1  # Number of Services
   MAX_OPERATIONS_INVOKED     = 10
   MAX_OPERATIONS_PERFORMED   = 10
   SERVICE_1                  = STANDARD_MR
```

In this case, the `[SCU_Or_SCP_Service_List]` supports the Standard MR SOP Class. For all services, it supports 10 maximum operations invoked and 10 maximum operations performed. When `MAX_OPERATIONS_INVOKED` and `MAX_OPERATIONS_PERFORMED` are not included in the service list, asynchronous communications are not negotiated. See a subsequent section for details on implementing DICOM asynchronous communications with Merge DICOM Toolkit.

Configuring Extended Negotiation for Clients

Extended Negotiation

Merge DICOM optionally supports configuration of DICOM Extended Negotiation information in service lists. Currently, the DICOM standard allows extended negotiation information for the Storage and Query/Retrieve Service classes as defined in PS3.4 of the standard. The extended negotiation information can be set for only the client (SCU). Server applications utilizing extended negotiation must set this information at run-time through the .NET Assembly.

```
[SCU_Service_List]
   SERVICES_SUPPORTED    = 2  # Number of Services
   SERVICE_1             = STUDY_ROOT_QR_FIND
   EXT_NEG_INFO_1        = 0x01
   SERVICE_2             = STUDY_ROOT_QR_MOVE
   EXT_NEG_INFO_2        = 0x01
```

In this case, the `[SCU_Service_List]` supports the Study Root Q/R Find and Move services. Both services have set a single byte of extended negotiation information set to hexadecimal 0x01. (In this case, this implies the Client supports relational Queries and Moves.) Multiple hexadecimal bytes can be set in the service list by listing each byte in the format "0x00 0x01 0x02".

Related General SOP Classes and Service Classes

DICOM Supplement 90 defines a mechanism in association negotiation to identify when a SOP Class is a customization of a generalized SOP Class. It also defines a method to identify the service class of a SOP Class that is proposed by an SCU. This allows flexibility in an SCP to support service classes for which it supports the generalized version of a SOP Class, but does not explicitly support the customized SOP Class. It also allows a mechanism to easily make an SCP that supports all storage service class SOP Classes that are proposed to it.

Related General SOP Classes can be supported in the application profile by defining a service list containing the related general SOP Classes for a given SOP class, and then assigning the service list to the SOP Class. The following example shows how this is done:

```
[SCU_SR_General]
    SERVICES_SUPPORTED    = 2
    SERVICE_1             = STANDARD_ENHANCED_SR
    SERVICE_2             = STANDARD_COMPREHENSIVE_SR

[SCU_DX_General]
    SERVICES_SUPPORTED    = 1
    SERVICE_1             = STANDARD_DX_PRESENT

[SCU_Service_List]
    SERVICES_SUPPORTED    = 3
    SERVICE_1             = STANDARD_BASIC_TEXT_SR
    REL_GENERAL_1         = SCU_SR_General
    SERVICE_CLASS_1       = 1.2.840.10008.4.2
    SERVICE_2             = STANDARD_IO_PRESENT
    REL_GENERAL_2         = SCU_DX_General
    SERVICE_CLASS_2       = 1.2.840.10008.4.2
    SERVICE_3             = STANDARD_CT
    SERVICE_CLASS_3       = 1.2.840.10008.4.2
```

In this case, the `SCU_SR_General` service list contains the related general SOP Classes for the `STANDARD_BASIC_TEXT_SR` service. The `REL_GENERAL_1` option points to the service list to use as the related general services for `SERVICE_1`.

The above example also shows how the service class can be defined for each SOP Class within a service list. For instance, `SERVICE_CLASS_3` in the above example specifies the service class for `SERVICE_3`. In this case, the UID for the Storage Service Class as defined in Supplement 90 is used.

The service lists above are only utilized by SCU applications. For SCP applications, there are several configuration options that define how Merge DICOM will negotiate an association when related general SOP Classes are included or the Service Class is included for a SOP Class.

When the `ACCEPT_STORAGE_SERVICE_CONTEXTS` configuration option is set to Yes, Merge DICOM will accept any proposed SOP class that is defined as supporting the Storage Service Class.

When the `ACCEPT_RELATED_GENERAL_SERVICES` configuration option is set to Yes, Merge DICOM will accept any SOP class proposed if the SCP supports in its service list any of the related general SOP Classes defined for a SOP Class proposed.

System Profile

The Merge DICOM System Profile (usually called `mergecom.pro`) contains configuration parameters for the DICOM Toolkit Library itself. The name and location of this file are specified in the `[MergeCOM3]` section of the Merge DICOM initialization file.

Many of these parameters should never need to be modified by the user, including low-level protocol settings such as time-outs. Only the parameters that should be understood by every user of the toolkit are discussed here; for a discussion of all parameters, see System Profile in Appendix D: Configuration Parameters.

Nothing works without the license number

Most importantly, you must place the license number you received when you purchased the toolkit in the `[ASSOC_PARMS]`section of the system profile. If the license you received with your toolkit was 83F3-3F26-FD6E you would need to set it in the `[ASSOC_PARMS]`section as follows:

```
[ASSOC_PARMS]
   LICENSE                       = 83F3-3F26-FD6E
   IMPLEMENTATION_CLASS_UID      = 2.16.840.1.113669.2.1.2
   IMPLEMENTATION_VERSION        = MergeCOM3_361
   ACCEPT_MUTLPLE_PRES_CONTEXTS = Yes
```

The toolkit sample applications, and your own applications that use the DICOM Toolkit Library will not work without a valid license number.

The above example of the [ASSOC_PARMS] section of the system profile also contains example implementation class UID and implementation version configuration values. The implementation class UID is intended by the DICOM standard to be unique for major revisions of an application entity. The implementation version is intended to be unique for the minor revisions of an application entity. These configuration values are used during association negotiation by Merge DICOM and are intended to aid in tracking versions of applications in the field.

The `ACCEPT_MULTIPLE_PRES_CONTEXTS` configuration value is used by server (SCP) applications. This value determines if multiple presentation contexts can be negotiated for a single DICOM Service. This option is discussed below.

As mentioned earlier, a listen port must be identified for your server AE. Port 104 is the standard DICOM listen port. This, along with the number of simultaneous TCP connection requests that can be queued up for acceptance (pending) for Merge DICOM toolkit, is specified in the `[TRANSPORT_PARMS]` section.

The `MAX_PENDING_CONNECTIONS` setting in the "mergecom.pro" file refers to the maximum number of outstanding connection requests per listener socket. The value of this configuration is passed by the toolkit to the listen() call on the socket as the backlog parameter and it specifies how many pending connections can be queued at any given time.

The `MAX_PENDING_CONNECTIONS` configuration option affects the accepting of associations but not the requesting of associations and it affects the behavior at the TCP level. In the default case, if more than five association requests arrive at once then only the first five will be accepted by TCP and passed to Merge DICOM Toolkit, the others would be refused at the TCP level.

```
[TRANSPORT_PARMS]
   TCPIP_LISTEN_PORT                        = 104
   # Max number of open listen channels
   MAX_PENDING_CONNECTIONS                  = 5
```

An important section of the System Profile is the `[MESSAGE_PARMS]` section:

```
[MESSAGE_PARMS]
   LARGE_DATA_STORE    = FILE #  | MEM Default = FILE
   LARGE_DATA_SIZE     = 200
   OBOW_BUFFER_SIZE    = 4096
   DICTIONARY_FILE     = /users/mc3adv/mrgcom3.dct
   TEMP_FILE_DIRECTORY = /users/mc3adv/tmp_files/
   MSG_INFO_FILE       = /users/mc3adv/mrgcom3.msg
```

Dealing with large data

The LARGE_DATA_STORE parameter informs the toolkit where it should store large data; either in memory, or in temporary files on disk. Large data is defined as a value for an attribute larger than LARGE_DATA_SIZE bytes. Pixel data associated with a medical image would most certainly be considered large data.

Performance Tuning

If you are running your process on a resource rich system that supplies plenty of physical and virtual memory, you should select `LARGE_DATA_STORE = MEM` to improve your performance. If your process is not so fortunate or you are dealing with messages with very large data values, you will want to use `LARGE_DATA_STORE = FILE.` In this case, the DICOM Toolkit will manage the large data in temporary files located in the `TEMP_FILE_DIRECTORY` you specify.

Callbacks MCattributeContainer

Large data that is of value representation OB (Other string of Bytes) or OW (Other string of 16-bit Words) or OF (Other string of 32-bit IEEE 754:1985 floating point words) is treated specially by the toolkit. Pixel Data, Curves, and Overlays are composed of this type of data. You can let the toolkit manage OB/OW/OF data for you like any other large data, or register your own Callback Class in your applications to deal with such data as it is being received or transmitted over the network. The use of Callbacks will be covered later when we discuss developing DICOM applications with the toolkit.

Performance Tuning

The `OBOW_BUFFER_SIZE` is used to tell the toolkit what size 'chunks' in bytes of OB/OW/OF data it should read in before either writing the data to a temporary file or passing it to your Callback Class. Choosing a large number for `OBOW_BUFFER_SIZE` means less time spent by your application process writing to temporary files or making callbacks, but results in a larger process size. If you need to use temporary files or callbacks, you should tune this parameter to maximize performance within the constraints of your runtime environment.

Message info Files

Another binary file supplied with the toolkit is the **message info file**. This file contains binary encoded message objects and is accessed when an application opens a message. Once open, these objects reside in memory, are 'filled in' by your application, and become a message object instance that can be exchanged over the network. The message info file, along with the data dictionary file, also make possible the powerful message validation capabilities of the DICOM Toolkit. The message info file is a binary file supplied with your toolkit with the default name of `mrgcom3.msg`. You also specify the location and name of the message info file using the `MSG_INFO_FILE` parameter.

Capturing Network Data

It is often useful to capture the raw data that is transmitted across the network to help determine exactly what each side of an association is sending. Network "sniffer" programs are often used to capture this data, but they are often not useful when the data is being transmitted over a secure network connection, as the data is often encrypted. Merge DICOM provides a network capture facility that will capture network data as it is sent or received. The data that is captured to one or more files and is formatted such that it can be analyzed using the MergeDPM© utility.

Refer to Appendix D for a discussion of the following configuration parameters that are used to configure the network capture facility. They are encoded in the [TRANSPORT_PARMS] section of the System Profile (mergecom.pro).

```
NETWORK_CAPTURE
CAPTURE_FILE
CAPTURE_FILE_SIZE
NUMBER_OF_CAP_FILES
REWRITE_CAPTURE_FILES
```

Service Profile

Items
MCitem

The Service Profile (usually called `mergecom.srv`) informs the toolkit what types of services and commands it supports, and what the corresponding message info files are. This file also lists the meta-services and **items** supported by the toolkit. Items are the nested 'sub-messages' contained within attributes of a message having the VR Sequence of Item (SQ) and will be discussed in greater detail later. The name and location of the service profile are specified in the `[MergeCOM3]` section of the Merge DICOM initialization file.

The service profile, along with the data dictionary and message info files, is generated from the Merge DICOM Database and should be modified by other means only by very experienced or specialized users.

Additional information about the service profile can be found in System Profile, Appendix D: Configuration Parameters.

Message Logging

Merge DICOM Toolkit supplies a message logging facility whereby three primary classes of messages can be logged to a specified file and/or standard output:

- Errors
- Warnings
- Status

Error messages include unrecoverable errors, such as "association aborted", or "failure to connect to remote application". Other error messages may be catastrophic but it is left to the application to determine whether or not to abort an association, such as an "invalid attribute value" or "missing attribute value" in a DICOM message.

Warnings are meant to alert toolkit users to unusual conditions, such as missing parameters that are defaulted or attributes having values that are not one of the defined terms in the standard.

Status messages give high-level messages describing the opening of associations and exchanging of messages over open associations.

As discussed earlier, other more detailed logging can be obtained by using the `T1_MESSAGE` through `T9_MESSAGE` logging levels. For example, the `T5_MESSAGE` logging level can be used to log the results of validate() or validateAttribute() methods of the MCdimseMessage class.

An excerpt from a Merge DICOM Toolkit message log file is included below that contains all three classes of messages: errors, warnings, and informational.

Message Log Example:

```
                                        .
                                        .
                                        .
(6196) 03-29 21:14:54.77 MC3 W: (0010,1010): Value from stream had problem:
(6196) 03-29 21:14:54.78 MC3 W: |   Invalid value for this tag's VR
(6196) 03-29 21:14:56.41 MC3(Read_PDU_Head) E: Error on Read_Transport call
(6196) 03-29 21:14:56.41 MC3(MCI_nextPDUtype) E: Error on Read_PDU_Head call
(6196) 03-29 21:14:56.41 MC3(Transport_Conn_Closed_Event) E: Transport
unexpectedly closed
(6196) 03-29 21:14:56.41 MC3(MCI_ReadNextPDV) I: DUL_read_pdvs error: UL
Provider aborted the association
                                        .
                                        .
                                        .
```

The first column contains the ID of the thread where the message was generated. The next column contains the date and the time when the message was generated.

The toolkit synchronizes the logging internally. Each call to log a message will block the calling thread until other pending calls are completed.

See *Using the Merge DICOM log file* section for more details on logging.

Utility Programs

The Merge DICOM Toolkit supplies several useful utility programs. These utilities can be used to help you validate your own implementations and better understand the standard.

All these utilities use the Merge DICOM Toolkit C Run-time Library and require that you set your `MERGE_INI` environmental variable to point to the proper configuration files (as described earlier).

mc3comp

Do a DICOM 'diff"

The `mc3comp` utility can be used to compare the differences between two DICOM objects. The objects can be encoded in either the DICOM file or "stream" format and do not have to be encoded in the same format. The utility will output differences in tags between the messages taking into account differences in byte ordering and encoding. The syntax for the utility is the following:

```
mc3comp [-t1 <syntax> -t2 <syntax>] [-e file] [-o -m1 -m2]
       file1 file2

t1 <syntax>   Optional specify transfer syntax of 'file1'
              message, where <syntax> = 'il' for implicit
              little endian (default), 'el' for explicit
              little endian, 'eb' for explicit big endian
t2 <syntax>   Optional specify transfer syntax of 'file2'
              message, where <syntax> = 'il' for implicit
              little endian (default), 'el' for explicit
              little endian, 'eb' for explicit big endian
e <file>      Optional exception file of all tags to
              ignore in comparison
o             Compare OB/OW/OF (e.g.,  binary pixel) data
m1            Compare 'file1' in DICOM-3 file format.
m2            Compare 'file2' in DICOM-3 file format.
h             Show these options.
file1         DICOM SOP Instance (message) file
file2         Another DICOM SOP Instance (message) file

Example:  mc3comp -t1 il -m2 -o 1.img 1.dcm
```

mc3conv

Convert Image Formats

The `mc3conv` utility can be used to convert a DICOM object between various transfer syntaxes and formats. The utility will read an input file and then write the output file in the transfer syntax specified in the command line. The utility can also convert between DICOM "stream" format and the DICOM file format. The syntax for the `mc3conv` utility is the following:

```
mc3conv input_file output_file [-t <syntax>] [-m] [-tag
       <tag> <"new value">]

input_file    DICOM SOP Instance (message) file
output_file   Output DICOM SOP Instance (message) file
t             Specify transfer syntax for 'output_file',
              where <syntax> = 'il' for implicit little
              endian (default), 'el' for explicit little
              endian, 'eb' for explicit big endian
m             Specify format of 'output_file' to be DICOM-
              3 media (Part 10) format.
tag           Change value for this tag in 'output_file',
              where <tag> = the tag that is to be changed
              in hex 0x... <new value> = the value for the
              tag in quotes ""
h             Show these options.

Example:  mc3conv in.img out.dcm -t el –m
```

mc3echo

Do a DICOM 'ping'

The `mc3echo` utility validates application level communication between two DICOM AE's. An echo test is the equivalent of a network 'ping' operation, but at the DICOM application level rather than the TCP/IP transport level.

All server (SCP) applications built with the DICOM Toolkit also have built-in support of the Verification Service Class and the C-ECHO command.

The command syntax follows:

```
mc3echo [-c count] [-r remote_host] [-l local_app_title]
       [-p remote_port] remote_app_title

c count       Integer number specifying the number of echoes
              to send to the remote host.  If -c is not
              specified, one echo will be performed
r remote_host       Host name of the remote computer  If -r
              is not specified, the default value for
              remote_host is configured in the Application
              Profile.
l local_app_title   Application title of this program.
              If -l is not specified, the default value
              for local_app_title is MERGE_ECHO_SCU
p remote_port       Port number the remote computer is
              listening on.  If -p is not specified, the
              default value for remote_host is configured
              in the Application Profile.
```

mc3list

Display Message Contents

`mc3list` displays the contents of binary DICOM message files in an easy to read manner. The message files could have been generated by `mc3file` (see below) or written out by your application.

`mc3list` is a useful educational tool as well as a tool that can be used for off-line display of the DICOM messages your application generates or receives.

The command syntax follows:

```
mc3list <filename> [-t <syntax>] [-m]

filename      Filename containing message to display
t             Specify transfer syntax of message, where
              syntax = "il" (implicit little endian), "el"
              (explicit little endian), or "eb" (explicit
              big endian)
m             Optional display a DICOM file object
```

If the DICOM service and/or command cannot be found in the message file, a warning will be displayed, but the message will still be listed.

The default transfer syntax is implicit little endian (the DICOM default transfer syntax). If the transfer syntax is incorrectly specified, the message will not be displayed correctly.

mc3valid

DICOM Message Validation tool

The `mc3valid` utility validates binary message files according to the DICOM standard and notifies you of missing attributes, improper data types, illegal values, and other problems with a message. `mc3valid` is a powerful educational and validation tool that can be used for the off-line validation of the DICOM messages your application generates or receives.

The command syntax follows:

```
mc3valid <filename> [-e | -w | -i] [-s <serv> -c <cmd>]
       [-l] [-m] [-q] [-t <syntax>]

<filename>    Filename containing message to validate
e             Display error messages only
w             Display error and warning messages (default)
I             Display informational, error, and warning
              messages
s <serv>      Optional force the message to be validated
              against service name "serv", used along with
              '-c'
c <cmd>       Optional force the message to be validated
              against command name "cmd", used along with
              '-s'
l             Optional list and select possible
              service-command pairs
m             Optional specify the input file as being a
              DICOM file object
q             Optional disable prompting for correct
              service-command pairs
t             Specify transfer syntax of message, where
              syntax = "il" (implicit little endian), "el"
              (explicit little endian), or "eb" (explicit
              big endian)
```

This command validates the specified message file; printing errors, warnings, and information generated to standard output. The user can force the message to be validated against a specified DICOM service-command pair if the message does not already contain this information.

If the service-command pair is not contained in the message, the program will list the possible service-command pairs and the user can select one of them. When using this program with a batch file, this option can be shut off with the -q flag.

The default transfer syntax is implicit little endian (the DICOM default transfer syntax). If the transfer syntax is incorrectly specified, the message cannot be validated.

limitations

While `mc3valid`'s message validation is quite comprehensive, it does have limitations. These limitations are discussed in detail in the description of the `validate` method of the MCdimseMessage class in the Assembly Windows Help File. The DICOM Standard should be always be considered the final authority.

mc3file

DICOM Message Generation Tool

Sample DICOM messages can be generated with the `mc3file` utility. You specify the service, command, and transfer syntax and `mc3file` generates a 'reasonable' sample message that is written to a binary file. The contents of this file are generated in DICOM file format or in exactly the format as the message would be streamed over the network.

The program fills in default values for all the required attributes within the message. You can also use this utility to generate its own configuration file, which you can then modify to specify your own values for attributes in generated messages.

These generated messages are purely meant as 'useful' examples that can be used to test message exchange or give the application developer a feel for the structure of DICOM messages. They are not intended to represent real world medical data.

The messages generated can be validated or listed with the mc3list and mc3valid utilities. The command syntax for mc3file is the following:

```
mc3file <serv> <cmd> <num> [-g <file>] [-c <file>] [-l] [-
      m] [-q] [-t <syntax>] [-f <file>]

<serv> <cmd>  These two options are always used together.
              They specify the service name and command
              for the message to be generated.  These
              names can be either upper or lower case.
              If the exact names for a service command
              pair are not known, the -l option can be
              used instead to specify the service name
              and command.  If the service name and
              command are improperly specified, mc3file
              will act as if the -l option was used and
              ask the user to input the correct service
              name and command.
<num>         This option specifies the number of message
              files to be generated by mc3file.  If the -
              g option is used, this option is not needed
              on the command line.  If the -c option is
              used, mc3file assumes the number is 1,
              although a higher number can be specified
              on the command line. mc3file will vary any
              fields that have a value representation of
              time when multiple files are generated,
              although when the -c option is used, the
              utility will use the time fields as
              specified in the configuration file.  Thus
              multiple message files generated with the -
              c option are identical.
g <filename>  This option causes mc3file to generate an
              ASCII configuration file.  The file
              contains a listing of all the valid
              attributes for the specified message.  The
              utility also adds sequences contained in
              the message along with their attributes.
              Each attribute in the file contains the
              tag, value representation, and the default
              value MC3File uses for the attribute.  If a
              given attribute has more than one value,
```

```
                    the character "\" is used to delimit the
                    values.  A default value listed as "NULL"
                    means the attribute is set to NULL.  If the
                    filename specified already exists, it will
                    be written over my MC3File.  The
                    configuration file can be modified and
                    reloaded into MC3File with the -c option to
                    generate a DICOM message.
c <filename>        This option reads in a configuration file
                    previously generated by mc3file.  The
                    service name and command for the message
                    need not be specified on the command line
                    because they are contained in <filename>.
                    Because multiple files generated with this
                    option are identical, mc3file assume only
                    one file should be generated.  This
                    assumption can be overridden by specifying
                    a number on the command line.
l                   This option lists all the service command
                    pairs supported by mc3file.  When
                    generating a message, this option can be
                    used instead of explicitly specifying the
                    service name and command on the command
                    line.  When specified alone in the command
                    line, the complete list of pairs is printed
                    out without pausing.
m                   This option allows the user to generate a
                    DICOM file.  When generating the  file
                    object, mc3file encodes the File
                    Meta Information.
q                   This option prevents mc3file from prompting
                    the user for correct service command pairs.
                    It is a useful option when running the
                    program from a batch file.
t <syntax>          This option specifies the transfer syntax
                    the DICOM message generated is stored in.
                    The default transfer syntax is implicit
                    little endian.  The possible values for
                    <syntax> are "il" for implicit little
                    endian, "el" for explicit little endian,
                    and "eb" for explicit big endian.
f <file>            This option allows the user to specify the
                    first eight characters of the names of the
                    DICOM message files being generated.
                    mc3file will then append a unique count to
                    the end of the filename for each message
                    being generated.  The default value is
                    "file" when creating a DICOM file and
                    "message" when creating the format that
                    DICOM messages send over a network.
```

MC3File retrieves default values for attributes from the text file "default.pfl". Unlike the "info.pfl" and "diction.pfl" files which are converted into binary files, "default.pfl" is used as a text file. It will first be searched for in the current directory and then in the message information directory. This file contains default values for all messages and for specific service-command pairs. This file can be modified to contain defaults specific for the user, although it is recommended that a backup of the original be kept. If this file is modified, there are no guarantees that the messages generated will validate properly.

Developing DICOM Applications

The Merge DICOM Toolkit .NET™ Assembly

The Merge DICOM Toolkit .NET Assembly provides classes and interfaces that represent all of the major components of the DICOM standard:

- Utility and Initialization methods -- **MC**
- DICOM constants
  - MCdicom
- DICOM Applications
  - Local applications -- **MCapplication**
  - Remote applications -- **MCremoteApplication**
- Merge DICOM Logging
  - MClog
  - MClogHandler
  - MClogInfo
  - MClogTime
- DICOM Associations - **MCassociation**
  - Association acceptors – **MCacceptor**
  - Association requesters -- **MCrequester**
  - Association negotiation
    - MCnegotiationInfo
      - McstorageNegotiation
      - MCqueryRetrieveNegotiation
  - MCproposedContext
  - MCproposedContextList
  - MCresultContext
  - MCtransferSyntax
  - MCtransferSyntaxList
- DICOM Messages and Message Elements
  - DIMSE messages
    - MCabstractMessage
      - MCdimseMessage
  - Data Elements -- **MCdataElement**
  - Data element identifiers – **MCtag**
  - DICOM Value Representation -- **MCvr**
  - Attributes – **MCattribute**
  - Attribute representations
    - Age String – **MCage**
    - Date – MCdate
    - DateTime – MCdateTime
    - Time – MCtime
    - Person Name – **MCpersonName**
    - UID – MCinstanceUID
    - Patient Name Component Group – **MCpnComponentGroup**
  - String Encoding -- **MCstringEncoder**
  - Attribute Collections - **MCattributeSet**
    - MCcommandSet
    - MCdataSet
    - MCitem
    - MCfileMetaInfo

- DICOM message handling callbacks
  - Data Sinks
    - MCdataSink interface
      - MCfileDataSink
      - MCmemoryDataSink
      - MCstreamDataSink
  - Data Sources
    - MCdataSource interface
      - MCfileDataSource
      - MCmemoryDataSink (MCmemoryDataSink implements the MCdataSink and MCdataSource interfaces.)
      - MCstreamDataSource
- DICOM message validation
  - MCvalidationError
  - **validate** and **validateAttribute** methods of MCdataSet, MCfile and MCdimseMessage classes
- DICOM Service class Information
  - MCsopClass
  - MCfailedSopInfo
  - MCrefSopInfo
  - MCrefStudyInfo
- DICOM Network Service classes - **MCdimseService**
  - MCbasicWorklistManagementService
  - MCpatientManagementService
  - MCprintManagementService
  - MCqueryRetrieveService
  - MCqueueManagementService
  - MCresultsManagementService
  - MCstorageService
  - MCstorageCommitmentService
  - MCstudyContentNotificationService
  - MCstudyManagementService
  - MCverificationService
- DICOM media services and objects
  - Files –
    - MCabstractMessage
      - MCfile
    - DICOMDIR – **MCdir**
    - DICOMDIR record - **MCdirRecord**
  - DICOM files service – **MCmediaStorageService**
- Exception handling – Mcexception, MCruntimeException and their sub-classes
- Compression related
  - Compression Interface –
    - MCcompression
      - RLE Compressor -- **MCrleCompressor**
      - RLE Decompressor -- **MCrleDecompressor**
      - JPEG and JPEG2000 Compressor -- **MCstandardCompressor**
      - JPEG and JPEG2000 Decompressor -- **MCstandardDecompressor**

This section of the User's Manual attempts to present the highlights of the Merge DICOM Toolkit Assembly in a logical manner as it might be used in real DICOM applications. The classes are presented in the context of example C# source code snippets, and alternative approaches are presented that tradeoff certain features for the benefits of increased performance.

Most of the discussions that follow pertain both to networking and media interchange applications; only the *Association Management, Negotiated Transfer Syntaxes,* and *Message Exchange* sections are networking specific. The last two sections; *DICOM Files* and *DICOMDIR* are media interchange specific.

Library Import

Visibility

In order to access the classes of the Merge DICOM Toolkit you should use the mergecom namespaces in your applications. This gives you visibility not only to the public and protected classes, but it also provides visibility to the constants contained in the MCdicom interface. The following namespaces are utilized by the Merge DICOM .NET assembly:

```
using Mergecom;
using Mergecom.Exceptions;
using Mergecom.Gen;
using Mergecom.Logging;
```

Library Constants

Constant values for DICOM tags

The **MCdicom** interface is generated from the Merge DICOM dictionary and contains constant values for all of the attributes defined by the DICOM standard. A copy of the interface source file is included with the library, although you will never have a need to actually compile the file.

Exception Handling

Each of the Toolkit classes is documented in the Merge DICOM .NET™ Assembly Help File. The exceptions that each class may throw are documented there.

Errors Happen

Merge DICOM methods throw exceptions derived from MCexception class that extend the System.ApplicationException class.

A special group of exceptions is MCruntimeException derived exception classes that are thrown for serious problems within the Assembly or when untenable conditions have been detected. Most applications would not catch these exceptions.

When an instance of MCexception is thrown, a message is logged to the Merge DICOM log file if the severity level of the exception is enabled in the merge.ini file. (Most exceptions have a severity level of "error" and are always logged.)

Exception Numbers

Each MCexception object has a public **exceptionNumber** field that identifies the exception. If desired, you may simply catch MCexception and then interrogate the exceptionNumber property to determine which specific exception was thrown. The exception numbers are defined by constants in the MCexception class and listed in Table 10. The following, for example, may be used to check exception after a network read.

```
try {
    msg = assoc.read(30000);
} catch (MCassociationReleasedStatus e) {
    //  Normal association completion:
    System.Console.Out.WriteLine("Association Released.");
} catch (MCexception e) {
    //  Association is no longer active
    if (e.exceptionNumber == NETWORK_SHUTDOWN)
        System.Console.Out.WriteLine("Network dropped");
    else if (e.exceptionNumber == ASSOCIATION_ABORTED)
        System.Console.Out.WriteLine("Assoc. abort");
    else if (e.exceptionNumber == INVALID_MESSAGE_RECEIVED)
        System.Console.Out.WriteLine("Invalid msg");
    else if (e.exceptionNumber == NETWORK_INACTIVITY_TIMEOUT)
        System.Console.Out.WriteLine("Network stopped");
    else {
        System.Console.Out.WriteLine("MCexception");
    }
    break;
} catch (System.Exception e) {
    System.Console.Out.WriteLine("Error on network read");
    assoc.abort();
    break;
}
```

You could, of course, have separate catch clauses for each exception.

Table 10: exceptionNumber property for each MCexception class

MCexception subclass	exceptionNumber property
MCalreadyExistsException	ALREADY_EXISTS
MCalreadyInitializedException	ALREADY_INITIALIZED_EXCEPTION
MCalreadyListeningException	ALREADY_LISTENING
MCassociationAbortedException	ASSOCIATION_ABORTED
MCassociationRejectedException	ASSOCIATION_REJECTED
MCassociationReleasedStatus	ASSOCIATION_RELEASED
MCattributeNotFoundException	ATTRIBUTE_NOT_FOUND
MCcallbackCannotComplyException	CALLBACK_CANNOT_COMPLY
MCcallbackInvalidArgumentException	CALLBACK_INVALID_ARGUMENT
MCconfigFileErrorException	CONFIG_INFO_ERROR
MCconfigurationError	CONFIGURATION_ERROR
MCconnectionFailedException	CONNECTION_FAILED
MCdicomdirException	DICOMDIR_ERROR
MCdisposedException	OBJECT_DISPOSED

MCexception subclass	exceptionNumber property
MCillegalArgumentException	ILLEGAL_ARGUMENT_EXCEPTION
MCinactivityTimeoutException	NETWORK_INACTIVITY_TIMEOUT
MCincompatibleValueException	INCOMPATIBLE_VALUE
MCinvalidEncodingWarning	INVALID_CHARS_IN_VALUE
MCinvalidEncodingException	INVALID_DIMSE_COMMAND
MCinvalidDirRecordTypeException	INVALID_DIR_RECORD_TYPE
MCinvalidLicenseInfoError	INVALID_LICENSE
MCinvalidMessageReceivedException	INVALID_MESSAGE_RECEIVED
MCinvalidTransferSyntaxException	INVALID_TRANSFER_SYNTAX
MClostConnectionException	LOST_CONNECTION
MCmaxOperationsExceededWarning	MAX_OPERATIONS_EXCEEDED
MCnegotiationAbortedException	NEGOTIATION_ABORTED
MCnetworkShutdownException	NETWORK_SHUTDOWN
MCnoAttributesException	NO_ATTRIBUTES
MCnoSuchRecordException	NO_SUCH_RECORD
MCnoSuchValueException	NO_SUCH_VALUE
MCnotFoundException	NOT_FOUND
MCnotIntiaiizedError	NOT_INITIALIZED_EXCEPTION
MCnotStandardElementException	NOT_A_STANDARD_DATA_ELEMENT
MCoperationNotAllowedException	OPERATION_NOT_ALLOWED
MCrequiredAttributeMissingException	REQUIRED_ATTRIBUTE_MISSING
MCtimeoutException	TIMEOUT
MCunacceptableServiceException	UNNACCEPTABLE_SERVICE
MCunknownHostNameException	UNKNOWN_HOST

Library Initialization

Before anything else

Your first call to the Assembly should be the static **mcInitialization** method of the MC class. Using the library without an explicit initialization results in automatic initialization using the default configuration. The **mcInitialization** method provides the location of the Initialization File (merge.ini) and allows Merge DICOM to perform essential startup tasks.

```
File mergeIniFile = new File("Path to merge.ini");
try {
   MC.mcInitialization(mergeIniFile);
} catch (MCalreadyInitializedException e) {
    …
} catch (MCinvalidLicenseInfoError e) {
    …
} catch (MCruntimeException e) {
    // DLL Not Found …
}
```

The mcInitialization method allows the Toolkit to perform the following critical processing.

The dynamic library Is loaded

First, the Merge DICOM C/C++ toolkit dynamic link library, **Mergecom.Native.dll** is loaded. The .NET CLR searches for the library using the platform's normal search path.

If the library cannot be located, an **`MCruntimeException`** exception will be thrown.

Configuration files located

After the dynamic library is loaded, the C toolkit is initialized. If an error occurs during initialization, the **`MCnotInitializedError`** exception is returned. This may include errors accessing the Merge DICOM Data Dictionary or Message info files. This could happen if you have not specified or incorrectly specified the `DICTIONARY_FILE` or `MSG_INFO_FILE` parameters in the System Profile (mergecom.pro). When this exception is thrown, further information on the reason for the exception may be contained in the merge.log file.

You must be licensed

Then the license key you specified in the System Profile (mergecom.pro) is checked for validity. If the key validation fails or the license key is not in the System Profile, an **`MCinvalidLicenseInfoError`** exception is thrown. Any further Library calls will result in an `MCnotInitializedError` runtime exception.

isInitialized

At any time you can check to see if the Library has been initialized by using the static `mcIsInitialized` method of the MC class. The method returns `true` if the `mcInitialization` method has already been called successfully.

Application Domains

Note that the Library can not be used from different application domains at the same time. In order to use it from a different application domain the Library must be released first in the application domain that initialized it.

Releasing the library

mcLibraryRelease

The static `mcLibraryRelease` method of the MC class is used to release the resources used by the Merge DICOM library. The method performs a graceful shutdown of the library. `mcInitialization` must be called again before using the library. This method is normally called before exiting a Merge DICOM application. This method will release all resources allocated by the Merge DICOM C/C++ Toolkit dynamic link library.

`mcLibraryRelease` must be called in the same application domain in which the `mcInitialization` call was made. After the Library is successfully released it can be re-initialized in either the same application domain or a new one.

Getting the Assembly Version

mcGetVersionString

You can use the static **mcGetVersionString** method of the MC class to retrieve the string identifying the Merge DICOM Library version. The library version number string is of the form "n.m.v" where n is major version number, m is minor version number and v is an interim release number.

Releasing Native Memory

Dispose

The Merge DICOM .NET Classes call a number of routines in the C Merge DICOM toolkit that allocate memory. The .NET classes have been written so native resources associated with a .NET class are automatically freed when the .NET CLR garbage collector cleans up the managed memory in the class. Note, however, that a number of classes implement the **`dispose`** method that an application can use if it wants greater control over when native memory is freed. Note that after this method is called for a specific instance, that instance can no longer be utilized by your application.

Here is a list of the classes that can be disposed explicitly:

- MCapplication
- MCassociation
- MCdataSet
- MCdimseMessage (contains a MCdataSet reference)
- MCfile (contains a MCdataSet reference)
- MCitem
- MCproposedContextList
- MCproposedContext
- MCtransferSyntaxList
- MCdata

Using the Merge DICOM log file

Logging a message

MClog

The Mergecom.Logging namespace contains definitions for utilizing the Merge DICOM log file (usually merge.log). The Merge DICOM logging mechanism allows logging at several different logging levels. Error, Warning, Info, and nine trace logging levels are allowed. The **MClog** class contains methods for logging to these various log levels. The method prototypes to log to several of these levels are:

```
public static void error(System.String msg)
public static void warning(System.String msg)
public static void info(System.String msg)
public static void t1(System.String msg)
public static void t9(System.String msg)
```

Each of these methods writes an entry into the Merge DICOM log file containing *msg*. The entry will be logged only if messages of specific type have been enabled in the merge.ini file.

The Merge DICOM log file is defined by the LOG_FILE parameter in the [MergeCOM3] section of the merge.ini file. To enable logging of specific types of messages, enter one or more of these parameters in the merge.ini file:

```
ERROR_MESSAGE =<destinations>
WARNING_MESSAGE=<destinations>
INFO_MESSAGE=<destinations>
```

`<destinations>` may be one or both of these values, separated by commas:

File to request that the messages be written to the LOG_FILE

Screen to request that the messages be written to the system's standard out

Memory to request that the messages be written to system memory. This option is useful if the application registers a custom log handler and no other destination is selected.

Please note that Error_Msg type messages are always written to the LOG_FILE.

Capturing Log Messages in Your Application

Log Callback

addHandler
removeHandler
MClogHandler
MClogInfo

You may want to capture log messages yourself, for example to integrate Merge DICOM log messages into your application's logging scheme.

To do this, create a new class that implements the **MClogHandler** interface. That interface requires you to provide a **receiveLogMessage** method that will be called by Merge DICOM whenever it is logging a message. Information about the logged message is passed to the **receiveLogMessage** method in an instance of the **MClogInfo** class.

You must register your log handler using the **addHandler** method of the **MClog** class. Once registered your **MClogHandler** class will be notified as messages are logged. Multiple handlers can be registered at any given time. You can de-register a handler by calling the **removeHandler** method of the **MClog** class.

Refer to the description of **addHandler** in the .NET Assembly Windows Help File, for more detailed information about controlling which messages will reach your handler.

If a log handler is registered the Library calls **receiveLogMessage** in the thread that generated the message but the calls are synchronized by the Library so the log handler does not have to deal with synchronization.

Registering Your Application

Create an application object
MCapplication getApplication

Before performing any network or media activity, your application must register its **DICOM Application Title** with the Merge DICOM Toolkit. This is done by calling the **`getApplication`** factory method of the **MCapplication** class. The **`getApplication`** method returns an **MCapplication** object that represents your DICOM Application Entity. Note that if the `getApplication` method is called more than once with the same argument, the same **MCapplication** instance is returned.

This DICOM Application Title is equivalent to the DICOM Application Entity Title defined earlier. If your application is a server, this application title must be made known to any client application that wishes to connect to you. If your application is a client, your application title may need to be made known to any server you wish to connect to, depending on whether the server is configured to act as a server (SCP) only to particular clients for security reasons.

For example, if your application title is "ACME_Query_SCP", you would register with the toolkit as follows:

```
MCapplication myAE;
myAE = MCapplication.getApplication("ACME_Query_SCP");
```

MCapplication objects can be disposed

If you wish to disable your application and free up its resources to the system you must release it using the **Dispose** method. This is necessary to free the resources used by the underlying native dynamic library.

```
myAE.dispose();
```

The Application Entity (AE) Title

AE titles should be Unique on a network

Current and potentially future DICOM service classes assume that Application Entity Titles on a DICOM network are unique. For instance, the retrieve portion of the Query/Retrieve service class specifies that an image be moved to a specific Application Entity Title (and not to a specific hostname and listen port). If two identical Application Entity Titles existed on a network, a server application can only be configured to move images to one of these applications. For this reason, the DICOM Application Entity Title for your applications should be configurable.

You can use the **ApplicationTitle** property to retrieve the Application Entity Title of the MCapplication object.

Association Management (Network Only)

Once you have registered one or more networking applications, you will probably want to initiate an association if you are a client, or wait for an association if you are a server. Clients will use the **`requestAssociation`** method of the **MCassociation** class and servers will use the **`startListening`** method of that class.

Preparing a Proposed Context List

MCproposedContextList
MCproposedContext

Before you establish an association connection you must determine what DICOM services you are prepared to handle and perhaps create an **MCproposedContextList** object to encapsulate the service information.

Using a Pre-configured Proposed Context List

Use a pre-configured service list …

If you wish to propose the services that are configured in a [<service_list_name>] section of the Application Profile (mergecom.app) file, you can use the **`MCproposedContextList.getObject`** factory method to retrieve an MCproposedContextList object based on the configured services.

```
MCproposedContextList myContext =
      MCproposedContextList.getObject("service_list_name");
```

Creating Your Own Proposed Context List

… or build your own service list at run-time

You also have the option of creating your own proposed context list at run time. An MCproposedContextList object represents a collection of **MCproposedContext** objects. Each MCproposedContext object represents:

- one DICOM service (SOP class)
- a set of transfer syntaxes you can support for the service
- a declaration of the roles you will play (SCU and/or SCP)

MCtransferSyntaxList
MCtransferSyntax

Before you can create an MCproposedContext object you must create one or more **MCtransferSyntaxList** objects representing the transfer syntaxes you want to use for the services. The MCtransferSyntaxList class represents a collection of **MCtransferSyntax** objects.

Using a Pre-configured Transfer Syntax List

MCtransferSyntaxList.
getObject

Again, you have the option of creating an MCtransferSyntaxList object based on configuration information, or you can create a new transfer syntax list at run time. To create an MCtransferSyntaxList object from transfer syntaxes configured in a [<syntax_list_name>] section of the Application Profile (mergecom.app) file, use the **`getObject`** factory method of the MCtransferSyntaxList class:

```
MCtransferSyntaxList mySyntaxes =
      MCtransferSyntaxList.getObject("syntax_list_name
      name");
```

Creating Your Own Transfer Syntax List

To create your own transfer syntax list you must create an array containing the predefined instances of the MCtransferSyntax objects you want to support, and call the constructor for the MCtransferSyntaxList class.

The MCtransferSyntax class contains a number of static properties which return MCtransferSyntax instances for each of the defined DICOM transfer syntaxes.

```
MCtransferSyntax[] mySynArray = new MCtransferSyntax[2];

mySynArray[0] = MCtransferSyntax.ExplicitBigEndian;
mySynArray[1] = MCtransferSyntax.JpegBaseline;

MCtransferSyntaxList myTSL = new
      MCtransferSyntaxList("MYSYNS", mySynArray);
```

Creating Your Own Proposed Context List

MCsopClass

For each service you want to include in your MCproposedContextList you must have an **MCsopClass** object. You can only get MCsopClass objects for services known to the Toolkit. You must either identify the service by its name, or you must identify it by its DICOM Unique Identifier (UID). Valid service names and UIDs are configured in the Service Profile (mergecom.srv) file.

```
MCsopClass myService1 =
      MCsopClass.getSopClassByName("STANDARD_CT");
MCsopClass myService2 = MCsopClass.getSopClassByUid
      ("1.2.840.10008.5.1.4.1.1.1");
```

For each service you want to include in your MCproposedContextList you must specify what roles your application is prepared to play for the service. Your application may indicate whether it is willing to perform the Service Class User(SCU) role and/or Service Class Provider(SCP) role. It may support either role or both roles. If these parameters are not specified the default role of the association requester is SCU only and the default role of the association acceptor is SCP only.

Now, you are ready to create an array of MCproposedContext objects that will be used in your new MCproposedContextList.

```
MCproposedContext[] myCtxArray = new MCproposedContext[2];
myCtxArray[0] = new MCproposedContext(myService1, myTSL);
bool scuRole = true, scpRole = true;
myCtxArray[1] = new MCproposedContext(myService2, myTSL,
      scuRole, scpRole);
```

Finally, you create your own MCproposedContextList:

```
MCproposedContextList myContextList = new
      MCproposedContextList("MYLIST1", myCtxArray);
```

MCproposedContext properties

AbstractSyntax
SCProle
SCUrole
ServiceName
TransferSyntaxList

The MCproposedContext class provides properties for the proposed context. The **`AbstractSyntax`** property retrieves the abstract syntax name associated with this proposed service. Note that this is equivalent to the SOP Class UID used to identify the DICOM service.

The **SCProle** property retrieves a code defining whether or not the application is willing to perform the SCP role, and the **SCUrole** property retrieves a code defining whether or not the application is willing to perform the SCU role.

The **ServiceName** property retrieves the name associated with this proposed service. This is the name configured in the mergecom.srv file.

The **TransferSyntaxList** property returns the MCtransferSyntaxList object which is a list of proposed transfer syntaxes for this service.

clearNegotiationInfo
setNegotiationInfo
contains
getContext
ListName
GetEnumerator
Size
toArray

MCproposedContextList properties

The MCproposedContextList class provides methods to retrieve properties of the proposed context list. The **clearNegotiationInfo** method clears any negotiation information that may have been set for a service. As a result, no negotiation information will be used for this service when the library attempts to establish an association with another DICOM application using this MCproposedContextList list. This method call is treated as a no-op if no negotiation information is registered for the service.

The **setNegotiationInfo** method is used to provide extended negotiation information for one or more services in the list. The extended negotiation information will be used during association negotiation by Merge DICOM Toolkit. The existence of extended negotiation information is dependent on the service and must be documented in the application's DICOM Conformance Statement. The negotiation information is provided by the toByteArray method of the specified MCnegotiationInfo instance. If a `null` pointer or an empty byte array is provided by the toByteArray method, this call is treated as if it were a clearNegotiationInfo method call.

The **contains** method determines if the proposed context list contains an MCproposedContext element or if it contains an element that has a specified service name.

The **getContext** method uses a service name to identify and return an MCproposedContext object from those encapsulated in the proposed context list.

The **ListName** property retrieves the name that uniquely identifies this list among all proposed context service lists used by the library.

The **GetEnumerator** method creates and returns an enumerator for all of the MCproposedContext objects encapsulated in the proposed context list. The iterator will present the elements in the order they were presented when this object was created.

The **Size** property contains the number of elements in the MCproposedContextList object.

The **toArray** method returns a reference to an array of MCproposedContext items that represent the proposed contexts used in the proposed context list.

MCresultContext properties

The MCresultContext class contains the properties of a service that has been accepted by both sides of a DICOM association. Instances of this class are returned by the FirstAcceptableContext and NextAcceptableContext properties of the MCassociation class. The NextAcceptableContext property is repeatedly accessed until it returns null, siginaling the end of the result context list.

getNegotiationInfo
PresentationContextID
ResultCode
RoleNegotiated
ServiceName
TransferSyntax

The get**NegotiationInfo** method retrieves any extended negotiation information that may have been received for the service. If none was received the method returns `false`.

If extended negotiation information was received for this service, the method returns `true` and calls the decode method of MCnegotiationInfo instance provided to provide the negotiation information.

The **PresentationContextID** property retrieves the DICOM Presentation Context ID assigned to this context's service for the current association.

The **ResultCode** property retrieves the result/reason code returned by the remote DICOM system for this context.

The **RoleNegotiated** property retrieves the role negotiated for the association requestor for this service. During association negotiation, for each proposed service, the association requestor proposes that it serve as an SCU (service class user) and/or an SCP (service class provider). An association acceptor can accept or reject the proposal.

The **ServiceName** property retrieves the Merge DICOM service name of this service which has been successfully negotiated between two DICOM application entities.

The **TransferSyntax** property retrieves a MCtransferSyntax object that contains the Merge DICOM id for the negotiated transfer syntax, as well as its DICOM Transfer Syntax UID.

MCtransferSyntax properties

==
Name
Uid
BigEndian
LittleEndian
Encapsulated
ExplicitVR

The MCtransferSyntax class provides methods to retrieve properties of the DICOM transfer syntax encapsulated by class instances.

Since the only instances are those defined by the static fields of the MCtransferSyntax class the '==' operator can be used to check two transfer syntax references for equality.

The **Name** property retrieves the transfer syntax name provided by Merge DICOM for the transfer syntax.

The **Uid** property retrieves the DICOM Transfer Syntax UID associated with the transfer syntax.

The `BigEndian` property determines if the transfer syntax uses big endian encoding or not, and the LittleEndian property determines if the transfer syntax uses little endian encoding or not.

The `Encapsulated` property determines if the transfer syntax uses encapsulation or not.

The `ExplicitVR` property determines if the transfer syntax explicitly specifies the DICOM Value Representation. Otherwise the VR is known implicitly according to each attribute tag.

MCtransferSyntaxList properties

contains
getSyntax
ListName
GetEnumerator
Size
toArray

The MCtransferSyntaxList class provides methods to retrieve properties of the DICOM transfer syntax list encapsulated by class instances.

The **`contains`** method determines if the list contains a specified transfer syntax. The **`getSyntax`** method retrieves a reference to a specific MCtransferSyntax object in the list.

The **`ListName`** property retrieves the name that uniquely identifies this list among all transfer syntax lists used by the library.

The **`GetEnumerator`** method creates and returns an Iterator for all of the MCtransferSyntax objects encapsulated in the list. The iterator will present the elements in the order they were presented when this object was created.

The **`Size`** property returns the number of elements in the list and the **`toArray`** method returns a reference to the encapsulated MCtransferSyntax object array.

Using Extended Negotiation Information

MCnegotiationInfo
MCstorageNegotiation
MCqueryRetrieve-
Negotiation

Some DICOM services allow you to use extended negotiation information during the association creation process. The Toolkit provides the **MCnegotiationInfo** abstract class to represent this process. Classes that extend the MCnegotiationInfo class must supply a **decode** method and a **toByteArray** method. The Library calls the decode method when it receives a buffer of extended negotiation information and it calls the toByteArray method to request that negotiation information be placed in a byte array for transmission.

Merge DICOM provides two sub-classes to the MCnegotiationInfo class: the **MCstorageNegotiation** class and the **MCqueryRetrieveNegotiation** class. (Refer to the Merge DICOM .NET™ Assembly Windows Help File for details.)

If you will be using extended negotiation, you will use the **setNegotiationInfo** method of the MCproposedContextList class to "register" your MCnegotiationInfo object. For example,

```
MCnegotiationInfo  myInfo;
MyContextList.setNegotiationInfo(myInfo);
```

Note that if you use the MCstorageService or MCqueryRetrieveService classes, this negotiation processing is handled for you automatically.

Starting an Association Requester

Association startup for a client application MCrequester

If your application will be an association requester, the MCapplication class provides several options you can use to request an association with a network partner.

Processing the association in the same thread

First, you must decide how you will process the new association. If your current thread will process the association, you will use the following form of the **MCassociation `requestAssociation`** method, providing the Remote Application Title of the server you wish to connect to through the use of the **MCremoteApplication** class:

```
MCapplication myAE =
       MCapplication.getApplication("ACME_Query_SCU");
MCremoteApplication remoteApp =
       MCremoteApplication.getObject("ACME_Query_SCP");
MCassociation myAssoc;
myAssoc = MCassociation.requestAssociation(myAE,
       remoteApp);
```

The getObject static method of **MCremoteApplication** allows you to load configuration information about the remote application from the Application profile. It is also possible to use the constructor from **MCremoteApplication** to supply all of the connection information for the remote application. See the Merge DICOM .NET Windows Help File description of MCremoteApplication for the format of these constructors.

Processing the association in the another thread

MCassociation

If you want a separate thread to process the association, you must first provide a class that will process the new association. The class must implement the **MCrequester** interface. That interface requires that your class have a **`start`** method that will be passed a newly-created **MCassociation** object. When Merge DICOM has successfully started an association, it will use a separate thread to call your class's `start` method. Everything you need to know about the association is provided in the MCassociation object. When your start method returns, Merge DICOM will end the thread.

In this case, you must not only supply the Remote Application Title of the server you wish to connect to, but you must also provide an instance of your MCrequester class:

```
class MyAssocHandler : MCrequester {
    …
    void start(MCassociation assoc) {
    …
    }
}

MyAssocHandler myHandler = new MyAssocHandler();
MCremoteApplication remoteApp =
       MCremoteApplication.getObject("ACME_Query_SCP");

MCassociation myAssoc;

myAssoc = myAE.requestAssociation("ACME_Query_SCP",
       remoteApp, myHandler);
```

If the `requestAssociation` call returns without throwing an exception the new thread has been started and your `myHandler` instance is called to process the association.

DEFAULT is to proposed the services defined in mergecom.app

Note that in the last two examples we only specified the Application Entity Title of the remote server. In this case the Library retrieves three important pieces of information from the information in the Application Profile (mergecom.app) file:

The list of services to be proposed is obtained from your application's entry in the Application Profile.

The name of the host the remote server is running on is obtained from the remote application entity's entry in the Application Profile.

The TCP/IP port number the remote server is listening on is obtained from the remote application entity's entry in the Application Profile.

Starting an Association Acceptor

Association startup for a server application

If your application will be an association acceptor (a server), the **MCassociation** class provides the **`startListening`** method to start up a thread that will listen for and process requests from remote DICOM Application Entities that wish to start an association with your local Application Entity.

MCacceptor MCassociation

Merge DICOM handles every association request that is received in a separate thread. You must supply an instance of a class that implements the **MCacceptor** interface. When the new association request is received, Merge DICOM calls the **start** method of your MCacceptor class instance to process the association, passing it a reference to a newly-created **MCassociation** object.

NOTE: the widely-known DICOM listen port is 104

Three forms of the `startListening` method exist. One allows you to specify which TCP/IP port is to be listened on; other will wait for association connections on the port specified by the TCPIP_LISTEN_PORT configuration parameter, or on port 104, if the TCPIP_LISTEN_PORT configuration parameter is missing from the System Profile (mergecom.pro). The third form allows an application to specify an IPEndPoint object representing the specific address and port number to listen on. This form also allows starting a 'dual-mode' listener that accepts both IPv4 and IPv6 connections, if the operating system supports such listeners.

Note that a given Application Entity may only make this call one time for a given address-port combination, without first calling stopListening (see below). It is possible though to start listeners on the same address and port for different application objects, in this case the toolkit will attach the existing listener to the second application.

Merge DICOM starts a separate thread for each listener.

In addition to the previous parametes, you must specify an MCproposedContextList object that describes the services your Application Entity is willing to support.

```
class MyAssocHandler : MCacceptor {
    …
void start(MCassociation assoc) {
        …
    }
}

MyAssocHandler myHandler = new MyAssocHandler();
MCapplication myAE = MCapplication.getApplication
       ("ACME_Query_SCP");
```

```
MCproposedContextList myContext =
       MCproposedContextList.getObject
       ("service_list_name");

int port;    // the port that is being listened on
```

To use the port number configured in mergecom.pro or the default port 104:

```
port = MCassociation.startListening(myAE, myContext,
       myHandler);
```

To listen on a specific port (e.g., 1114):

```
port = MCassociation.startListening(myAE, 1114, myContext,
       myHandler);
```

To listen on a specific network interface and port number:

```
IPEndPoint ep = new IPEndPoint( localAddress, 1114 );
Mcassociation.startListening( scpApp, ep, false, myContext,
       myHandler );
```

STOP!

stopListening

If your application wants to stop listening for association requests on a given port, it must call the **`stopListening`** method of the MCassociation class. This simply requests that the library no longer accept connection requests on the port that are directed to this Application Entity. A server program may call this when it is about to shut down, and then wait for any active threads to finish.

```
MCapplication myAE =
       MCassociation.getApplication("ACME_Query_SCP");
…
MCassociation.stopListening(myAE, port);
```

Accepting or Rejecting the Association

Before DICOM messages can be exchanged across the association, the association acceptor must either accept or reject the association request from the association requestor.

MCresultContext
NumberOfAcceptable-Contexts
FirstAcceptableContext
NextAcceptableContext
MCassociation
accept
reject
MCresultContext

When the Merge DICOM library calls the `start` method of your MCacceptor class, it has already determined that both the local and remote applications wish to perform at least one common service. The **`FirstAcceptableContext`** and **`NextAcceptableContext`** properties of the MCassociation class may be used to examine the services that are agreeable to both sides. The **`NumberOfAcceptableContexts`** property retrieves the number of contexts that is acceptable to both sides. Each of these calls returns an **MCresultContext** object that can be interrogated to determine the properties of each acceptable service. The **`NextAcceptableContext`** property can be called multiple times to traverse through the acceptable contexts. A null will be returned by the property when the end of the list has been reached. The **`FirstAcceptableContext`** method can be called to reset the list and traverse through it again. Several other methods are available in MCassociation class to inquire about the proposed association. (Please refer to the MCassociation and MCresultContext classes in the Windows Help File.)

Note that many applications don't have a need to call the `FirstAcceptableContext` or `NextAcceptableContext` methods since it is acceptable that any of the services it negotiated were agreeable to both sides.

If this application agrees with the acceptable services, it calls the **`accept`** method of `MCassociation` to establish an association between the two applications. If it disagrees, for some reason, it calls the **`reject`** method.

The acceptor decides

```
class MyAssocHandler : MCacceptor {
    …
    void start(MCassociation assoc) {
        …
        try {
            MCresultContext ac = assoc.FirstAcceptableContext;
        while (ac != null) {
            // Insert your check here
            ac = assoc.NextAcceptableContext;
        }
        if (<the services negotiated are acceptable>)
            assoc.accept();
        else
            assoc.reject();
        } catch (MCassociationAbortedException e) {…}
    }
}
```

Accept or Reject

If you are rejecting the association, DICOM allows you to specify the reason you are rejecting and what type of rejection it is. If you specify no parameters to the `reject` method (as in the example above), it is assumed that the reject is permanent (i.e. there is no need for the remote application to "call later") and no reason is provided.

If you wish to give a reason, use this form of the `reject` method:

```
bool permanentReject = false;
assoc.reject(permanentReject,
       MCrejectReason.TEMPORARY_CONGESTION);
```

The reason codes are defined in the `MCrejectReason` enumerated value. These codes are available:

```
MCrejectReason.NO_REASON_GIVEN
MCrejectReason.APPLICATION_CONTEXT_NAME_IS_NOT_SUPPORTED
MCrejectReason.CALLING_AE_TITLE_NOT_RECOGNIZED
MCrejectReason.CALLED_AE_TITLE_NOT_RECOGNIZED
MCrejectReason.TEMPORARY_CONGESTION
MCrejectReason.LOCAL_LIMIT_EXCEEDED
```

Requestor reacts

The association requestor (normally the client application) must check for association rejection when it makes the association request:. Other exceptions also need to be checked (please see to the MCaassocition class in the Assembly Windows Help File).

```
try {
    myAssoc = MCassociation.requestAssociation(myAE,
        remoteApp);
} catch (MCassociationRejectedException e) { … }
    catch (MCconnectionFailedException e) { … }
    catch (MCnegotiationAbortedException e) { … }
    catch (MCunknownHostNameException e) { … }
    catch (Exception e) { … }
```

Negotiated Transfer Syntaxes

Dealing with transfer syntaxes

Merge DICOM Toolkit supports all currently approved standard and encapsulated DICOM transfer syntaxes. Encapsulated transfer syntaxes require compression of the pixel data contained in the message. These messages can be sent and received by the toolkit. A subsequent section describes how compression and decompression can be done with the library. Encoding of this pixel data is also discussed below.

For DICOM Toolkit users, the toolkit allows for the negotiation of more than one transfer syntax for a given DICOM service. This functionality is of most use for applications supporting encapsulated transfer syntaxes. This functionality may be disabled by use of the `ACCEPT_MUTLPLE_PRES_CONTEXTS` configuration value. In order to understand how it is implemented, a more in depth description of DICOM association negotiation is required.

During association negotiation a client (SCU) application will propose a set of presentation contexts over which DICOM communication can take place. Each presentation context consists of an abstract syntax (DICOM service) and a set of transfer syntaxes that the client (SCU) understands. The server (SCP) will typically accept a presentation context if it supports the abstract syntax and one of the proposed transfer syntaxes.

As previously discussed, the abstract and transfer syntaxes supported by a server (SCP) are defined through a service list contained in the Merge DICOM Application Profile. When support within a server (SCP) is limited to the three non-encapsulated DICOM transfer syntaxes, the toolkit will transparently handle the use of multiple presentation contexts for a DICOM service. However, when encapsulated DICOM transfer syntaxes are used, the server (SCP) must be able to determine the transfer syntax of messages it receives so that it can properly parse the pixel data contained in them. When a single presentation context is negotiated for a DICOM service, the **`FirstAcceptableContext`** and **`NextAcceptableContext`** MCassociation properties can be used to determine the transfer syntax for a service.

When more than one presentation context is negotiated for a service, the **`TransferSyntax`** property of the MCdimseMessage class must be used to set or get this transfer syntax. The following is a typical call to this method:

```
MCdimseMessage dm;  // A DICOM message just received
MCtransferSyntax ts = dm.TransferSyntax;
```

Exchange of messages over the network is discussed further below.

Transfer Syntax Lists for SCUs

The presentation contexts supported for client (SCU) applications using Merge DICOM are also defined through the Merge DICOM Application Profile. The following is a typical client's (SCU) configuration:

```
[Acme_Store_SCP]
      PORT_NUMBER          = 104
      HOST_NAME            = acme_sun1
      SERVICE_LIST         = Storage_Service_List

[Storage_Service_List]
      SERVICES_SUPPORTED   = 1  # Number of Services
      SERVICE_1            = STANDARD_CT
```

In this case, the client (SCU) would propose the CT Image Storage service in a single presentation context. The transfer syntaxes for each service are the three standard (non-encapsulated) DICOM transfer syntaxes.

The following example is the configuration for a client (SCU) that supports more than one presentation context for a service:

```
[Acme_Store_SCP]
       PORT_NUMBER          = 104
       HOST_NAME            = acme_sun1
       SERVICE_LIST         = Storage_Service_List

[Storage_Service_List]
       SERVICES_SUPPORTED   = 2  # Number of Services
       SERVICE_1            = STANDARD_CT
       SYNTAX_LIST_1        = CT_Syntax_List_1
       SERVICE_2            = STANDARD_CT
       SYNTAX_LIST_2        = CT_Syntax_List_2

[CT_Syntax_List_1]
       SYNTAXES_SUPPORTED   = 1  # Number of Syntaxes
       SYNTAX_1             = JPEG_BASELINE

[CT_Syntax_List_2]
       SYNTAXES_SUPPORTED   = 1  # Number of Syntaxes
       SYNTAX_1             = IMPLICIT_LITTLE_ENDIAN
```

If a server (SCP) accepts both of these presentation contexts, the client (SCU) must use the **`TransferSyntax`** property of the MCdimseMessage class to specify which presentation context to send a message over as follows:

```
MCdimseMessage dm;  // A DICOM message ready to send
dm.TransferSyntax = MCtransferSyntax.JpegBaseline;
```

Transfer Syntax Lists for SCPs

Server (SCP) applications are configured differently than client (SCU) applications. An SCP should include all of the transfer syntaxes a service supports in a single transfer syntax list. If more than one transfer syntax list is used for a service, server (SCP) applications will only support the transfer syntaxes contained in the first transfer syntax list. The following is an example configuration for a server (SCP):

```
[Storage_Service_List]
       SERVICES_SUPPORTED   = 1  # Number of Services
       SERVICE_1            = STANDARD_CT
       SYNTAX_LIST_1        = CT_Syntax_List_SCP

[CT_Syntax_List_SCP]
       SYNTAXES_SUPPORTED   = 4  # Number of Syntaxes
       SYNTAX_1             = JPEG_BASELINE
       SYNTAX_2             = EXPLICIT_LITTLE_ENDIAN
       SYNTAX_3             = IMPLICIT_LITTLE_ENDIAN
       SYNTAX_4             = EXPLICIT_BIG_ENDIAN
```

As discussed previously, for server (SCP) applications, the order in which transfer syntaxes are specified in a transfer syntax list dictates the priority Merge DICOM places on them during association negotiation. In this case, Merge DICOM would select JPEG_BASELINE if proposed, followed by EXPLICIT_LITTLE_ENDIAN, IMPLICIT_LITTLE_ENDIAN, and EXPLICIT_BIG_ENDIAN.

Network message exchange is discussed further in one of the following sections.

Merge DICOM Message Classes

A brief review
MCdataElement
MCtag
MCattribute
MCvr

Before we discuss the process of transferring messages, we must discuss some basic Merge DICOM classes used to represent a DICOM message.

DICOM data elements (**MCdataElement** class) are identified by a unique number (**MCtag** class). A DICOM attribute (**MCattribute** class) contains the value of a DICOM data element. An attribute has assigned to it a value representation (**MCvr** class), a value multiplicity(n[-n]) and a value type (1, 1C, 2, 2C, 3).

MCdimseMessage contains an MCcommandSet and, optionally, an MCdataSet

DICOM messages sent across a network connection on an association are represented by the **MCdimseMessage** class. DICOM supports different sets of attributes (**MCattributeSet** class). A DIMSE message contains a command set **MCcommandSet** class) containing header information used by the DICOM DIMSE service, plus, optionally, a data set (**MCdataSet** class) containing DICOM data being exchanged. The value of a Sequence of Items (SQ) attribute is zero or more DICOM items, where each item is a set of attributes. These DICOM items are represented by the **MCitem** class. (See figure below)

SQ attributes have MCitem objects as values

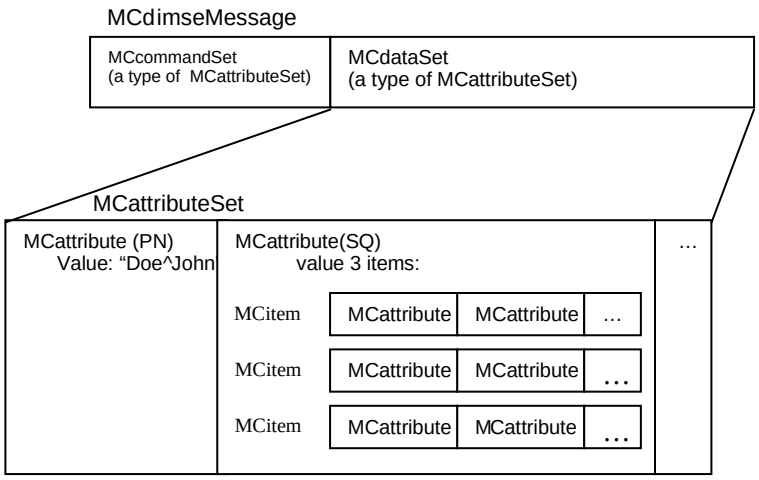

Figure 11: DIMSE messages and attribute sets

Association Message Handling

Once an association has been negotiated, the two cooperating applications, the Service Class Provider (SCP) and the Service Class User (SCU), exchange DICOM messages on the association network connection. These messages are encapsulated in MCdimseMessage objects.

Applications use instances of the **MCdimseService** class (or one of its sub-classes) to send request and reply messages, and use the **`read`** method of the **MCassociation** class to retrieve the messages sent by the network partner.

Send messages using one of the MCdimseService sub-classes or the MCdimseService class itself

For example, if an application is using the DICOM Storage Service, it would construct an instance of the **MCstorageService** and then use its **`sendStoreRequest`** method to send a request message and use, for example, its **`sendSuccessResponse`** method to reply to a received DIMSE message. When you construct a new MCdimseService class you provide the MCassociation object that the service is to operate on.

```
MCstorageService myService = new MCstorageService(assoc);
MCdataSet ds = new MCdataSet(C_STORE_RQ, "STANDARD_CR");
// encode the ds
   …
String affectedSopClassUid = "1.2.840.10008.5.1.4.1.1.1";
MCdimseMessage dm;
try {
   // This call creates a new McdimseMessage,
   // using the MCdataSet(ds) provided.
   // It then sends the message to the network partner.
   dm =
   myService.sendStoreRequest(ds, affectedSopClassUid);
} catch (Exception e) { … }
```

Note: The example above is just a sample of the methods available with the MCstorageService class. Refer to the Assembly Windows Help File for complete details on using that class.

Read messages using MCassociation.read

The MCassociation **`read`** method to retrieves the next message sent by the network partner. The `read` method returns an instance of the **MCdimseMessage** class. See the section *Negotiated Transfer Syntaxes*.

Note that DICOM requires that one or more reply message be sent in response to all DIMSE messages received, depending on the DICOM service being performed.

```
MCdimseMessage dm;
try {
   long timeout = 30000;  // 30 second timeout
   dm = assoc.read(timeout);
   if (dm == null)
      // Timeout!
   // The MCdimseMessage just sent is returned
} catch (Exception e) { … }
```

The current implementation of **`read`** method does not support timeout values that are less than one second. Any value that is less than 1000 and greater than zero will be rounded to 1000 (one second). Subsequently any value greater than 1000 is rounded to the nearest thousand (second).

Releasing or Aborting the Association

The DICOM standard requires the association requester to release the association when no further processing is required. This is done using the **`release`** method of the MCassociation class.

```
assoc.release();
```

At any time either association partner may abort the association. This is used only in abnormal situations.

```
assoc.abort();
```

After calling `release` or `abort`, no other methods should be called for the association object.

When the network partner releases or aborts the association, the other application is notified by an exception thrown by the read method.

```
MCdimseMessage dm;
try {
    long timeout = 30000;  // 30 second timeout
    dm = assoc.read(timeout);
    if (dm == null)
        // Timeout!
    // The MCdimseMessage just sent is returned
}   catch (MCassociationReleasedStatus e) { … }
    catch (MCassociationAbortedException e) { … }
    catch (MCexception e) { … }
```

Association Properties

The MCassociation class contains several properties that can be used to retrieve information about the association.

Application Context Name

ApplicationContextName

The **`ApplicationContextName`** property contains the name of the application context in use by the local and remote applications on this association connection. It is in the form of a DICOM Unique Identifier.

An application context is an explicitly defined set of application service elements, related options, and any other information necessary for the interworking of application entities over the association.

Currently there is only one DICOM Application Context Name that is defined for the DICOM standard: "1.2.840.10008.3.1.1.1"

Refer to Annex A in Part 7 of the DICOM standard for more information.

TCP/IP Listen Port

ListenPort

The **`ListenPort`** property retrieves the port number that the association is using to listen for TCP/IP connection requests.

MCapplication object of the local AE

LocalApplication

The **`LocalApplication`** property retrieves the MCapplication object identifying the DICOM application responsible for this MCassociation object.

Application Entity Title

LocalApplicationTitle
RemoteApplicationTitle

Each DICOM application is assigned a application entity ID, known also as the application title. The **`LocalApplicationTitle`** property retrieves the application title of the local application and the **`RemoteApplicationTitle`** property retrieves the application title of the remote application

Implementation Class UID and Implementation Version

LocalImplementationClassUid
LocalImplementationVersion
RemoteImplementationClassUid
RemoteImplementationVersion

The identification of an implementation of the DICOM standard relies on two pieces of information: the Implementation Class UID (required) and the Implementation Version Name (optional). The DICOM standard requires that association requestors and acceptors notify each other of their respective Implementation Class UID. The **`LocalImplementationClassUid`** property returns the Implementation Class UID of the local application and the **`LocalImplementationVersion`** property returns the Implementation Version of the local application. The **`RemoteImplementationClassUid`** property returns the Implementation Class UID of the remote application and the **`RemoteImplementationVersion`** property returns the Implementation Version of the remote application.

Maximum PDU Sizes

LocalMaxPDUsize
RemoteMaxPDUsize

During association negotiation Merge DICOM and the remote DICOM system exchanged the maximum size of Protocol Data Units that each is willing to receive. Each system commits to send TCP/IP data no larger than that negotiated for the receiver. The **`LocalMaxPDUsize`** property returns the size of the largest PDU that the local system is willing to receive. The **`RemoteMaxPDUsize`** property returns the size of the largest PDU that the remote system is willing to receive.

The Proposed Context List

ProposedContextList

NumberOf-
ProposedContexts

When an MCassociation object is constructed, an MCproposedContextList object is usually provided to define the services that the local application is willing to perform. If no MCproposedContextList was provided, Merge DICOM created an MCproposedContextList from information defined in the mergecom.app file. The **`ProposedContextList`** property returns a reference to the MCproposedContextList object used.

During association negotiation the DICOM association requestor proposes the services it wishes to use. The association acceptor can then reject or accept each of the proposed services**.** The **`NumberOfProposedContexts`** property returns the number of services proposed by the association requestor.

The Read Timeout Value

ReadTimeout

The **`ReadTimeout`** property returns the timeout value specified by the *timeout* parameter of the last read method call for this association. If no read has been called yet, zero (0) will be returned.

The Remote Host's Name and Address

RemoteHostName
RemoteIpAddress
RemotePort

Merge DICOM applications communicate with each other using TCP/IP. The **`RemoteHostName`** property returns the IP name of the remote application's host computer and the **`RemoteIpAddress`** property returns the IP address of the remote application's host computer and the **`RemotePort`** property retrieves the port number that the remote system is using to listen for TCP/IP connections.

Association Role

Acceptor
Requester

MCassociation objects may be constructed as acceptors (those waiting for and responding to DICOM association requests) or as requesters (those making DICOM association requests). The **`Acceptor`** property returns `true` if this MCassociation object represents an acceptor association and the **`Requester`** property returns `true` if this MCassociation object represents an requester association.

Association State

For a DICOM association requester application, an association is considered active from the moment the requester receives a successful return from the MCapplication requestAssociation call until it calls the MCassociation release method or until the association is aborted by either the local requester or remote acceptor application.

For a DICOM association acceptor application, an association is considered active from the moment the acceptor calls its accept method until the association is released by the remote requester or until the association is aborted by either the local acceptor or remote requester application.

Active

The **`Active`** property returns true if the association is currently active. Refer to the MCassociation **`Active`** property description in the Assembly Windows Help File for more detailed information about association states.

Using the MCsopClass class

All about SOP classes

getSopClassByName
getSopClassByUid

The MCsopClass class encapsulates the properties of a DICOM service. Merge DICOM manages instances of the MCsopClass for each service defined in the System Profile (mergecom.srv). MCsopClass class has two static methods, **getSopClassByName** and **getSopClassByUid**, which can be used to retrieve these Merge DICOM managed instances. These two methods take either the service name, or a DICOM SOP Class UID. Both of these are defined in the System Profile (mergecom.srv).

BaseClasses
Commands
Name
Number
Uid
BaseClass
MetaClass
Equals

The MCsopClass has the following get methods: **BaseClasses, Commands, Name, Number, Uid, BaseClass, MetaClass**, plus an overridden **Equals** method. See the sample code below.

```
MCsopClass sop1 = MCsopClass.getSopClassByUid
      ("1.2.840.10008.5.1.1.1");
MCsopClass sop2 = MCsopClass.getSopClassByName
      ("BASIC_FILM_SESSION");
if (!sop1.Equals(sop2))
   System.Console.Out.WriteLine("They should be equal");
System.Collections.IList baseClasses = sop1.BaseClasses;
```

```
if (baseClasses != null)
   System.Console.Out.WriteLine("? Not a meta sop");
System.Collections.BitArray commands = sop1.Commands;
if ( !sop1.Name.Equals( sop2.Name ) )
   System.Console.Out.WriteLine("? Should be equal");
if ( !sop1.Uid.equals( sop2.Uid ) )
   System.Console.Out.WriteLine("? Should be equal");
if ( sop1.Number != sop2.Number )
   System.Console.Out.WriteLine("? Should be equal");
if ( sop1.BaseClass )
   System.Console.Out.WriteLine("That's right");
if ( !sop1.MetaClass )
   System.Console.Out.WriteLine("That's right");
```

Earlier we saw how MCsopClass instances are used when creating new MCproposedContext objects.

Using the MCvr class

The **MCvr** class encapsulates a DICOM Value Representation (VR). DICOM defines a set of valid value representations for data elements encoded to the standard. The MCvr class contains a number of staticly defined MCvr instances for each of the valid VRs. The static field names are:

MCvr name	Value Representation used for …
vrAE	Application Entity
vrAS	Age String
vrAT	Attribute Tag
vrCS	Code String
vrDA	Date
vrDS	Decimal String
vrDT	Date Time
vrFD	Floating Point Single
vrFL	Floating Point Double
vrIS	Integer String
vrLO	Long String
vrLT	Long Text
vrOB	Other Byte String
vrOF	Other Float String
vrOL	Other Long String (Note: this was a VR defined in a draft supplement for DICOM that was never adopted.)

MCvr name	Value Representation used for …
vrOW	Other Word String
vrPN	Person Name
vrSH	Short String
vrSL	Signed Long
vrSQ	Sequence of Items
vrSS	Signed Short
vrST	Short Text
vrTM	Time
vrUI	Unique Identifier
vrUL	Unsigned Long
vrUN	Unknown
vrUS	Unsigned Short
vrUT	Unlimited Text

Several Merge DICOM class methods make use of MCvr class references. Most often, you will be using one of the static values in the MCvr class, which encapsulate each of the standard DICOM Value Representations.

MCvr

ToString
validateValue

The MCvr class provides some convenient methods. The **`ToString`** method is overridden to return a 2-character string that represents this Value Representation. For example, using the MCvr field vrSQ, vrSQ.toString returns "SQ".

The **`validateValue`** method can be used to validate the encoding of an attribute according to the rules defined in DICOM.

A number of other properties are also defined for MCvr that describe the properties of a VR. These properties are detailed in the Windows Help file.

Using the MCtag class

MCtag

An **MCtag** object identifies a DICOM attribute. All class methods that require an attribute identifier use the MCtag object for that identification. Note that in most cases these routines also allow the use of uint values to represent a DICOM tag.

As mentioned before, a DICOM tag is usually written as an ordered pair of two byte numbers. The first two bytes are sometimes called a **group number**, with the last two bytes being called an **element number** (e.g., (0010, 0010), (0038, 001C), ...).

The MCtag class addresses the fact that DICOM allows both private and non-private attributes. The group number for private attributes must always be odd, while the group number for non-private attributes must always be even. Private attributes belong to a private group, identified by a private code string, and private groups may only have 254 elements, numbered 1 through 255.

Constructing non-private tags

Non-Private Tags

Non-private tags may be constructed using a 32-bit integer number, or using a 16-bit group number plus a 16-bit element number.

```
MCtag tag;
try {
   tag = new MCtag(0x00080010);
} catch (MCillegalArgumentException e) {}

or-

try {
   tag = new MCtag(0x0008, 0x0010);
} catch (MCillegalArgumentException e) {}
```

The group and element numbers may be specified as two unsigned integers. An MCillegalArgumentException will be thrown if the group number portion of the tag number is odd.

Private Tags

Constructing private tags

Tags for private attributes are constructed by providing a private group code string in addition to the private group number and element number.

```
MCtag tag;
try {
   tag = new MCtag("Group1", (uint)0x0009, (uint)0x10);
} catch (MCillegalArgumentException e) {}

or-

try {
   tag = new MCtag( 0x00091010U, "Group1" );
} catch (MCillegalArgumentException e) {}
```

The numbers may be specified as two unsigned integers. An MCillegalArgumentException will be thrown if the group number was an even number, if the element number was greater than 0xFF, or if the private group code string was empty.

Using the MCdataElement class

MCdataElement

The MCdataElement class is used to define DICOM data elements. This is most often used to define data elements that are not defined in the data dictionary. DICOM data elements may be "standard" (i.e. they are defined in the data dictionary) or "non-standard" (those not defined in the data dictionary).

Merge DICOM allows you to define both private and non-private data elements, by simply using a private or non-private MCtag object to identify the data element.

Constructing standard data elements

Standard Data Elements

Standard data elements are built using the following constructor.

```
MCdataElement de;
try {
    MCtag tag = new MCtag(0x0008, 0x0010);
de = new MCdataElement(tag);
} catch (MCnotStandardElementException e) {}
```

An `MCnotStandardElementException` will be thrown if the data element is not defined in the data dictionary.

Non-standard Data Elements

Constructing non-standard data elements

Since non-standard data elements are by definition not in the data dictionary, you must supply the properties of data elements that are not recorded in the data dictionary. These properties must be defined:

- The **`name`** of the data element. If `name` is not provided, the attribute will be named "<UNKNOWN>" by the library. If provided, `name` must not be null and must have a length between 0 and 30. If longer than 30 it will be truncated. If 0 (i.e. ""), the name will be reported as blank in reports.

- The ***vr*** parameter specifies the Value Representation of the data element. It must be one of the MCvr instances that are static members of the MCvr class. An exception will be thrown if an invalid MCvr reference is provided. These pre-defined MCvr object references are available to use:

  vrAE, vrAS, vrCS, vrDA, vrDS, vrDT, vrIS, vrLO, vrLT, vrPN, vrSH, vrST, vrTM, vrUT, vrUI, vrSS, vrUS, vrAT, vrSL, vrUL, vrFL, vrFD, vrUN, vrOB, vrOW, vrOL, vrSQ

- The data element's Value Multiplicity is specified by the ***n*** and ***m*** parameters. The following table describes the effect of specifying zero, one or both of the *n* and *m* parameters:

n specified	*m* specified	Value Multiplicity is
no	no	1-many (no upper limit)
yes	no	must have exactly *n* values
yes	yes	*n* – *m* (at least *n* values; no more than *m* values) Use Short.MAX_VALUE to specify no upper limit.

Non-standard data elements are built using the following constructor.

```
MCdataElement de;
MCvr vr = vrDT;
ushort n = 1;// minimum number of values allowed
ushort m = 5;  // maximum number of values allowed
String name = "My Private Data Element";
```

```
try {
    MCtag tag1 = new MCtag(0x8014, 0x0010);

    de = new MCdataElement(vr, tag1, n, m, name);
}   catch (MCillegalArgumentException e) {}
```

An MCillegalArgumentException will be thrown if vr parameter was not a valid MCvr reference.

An MCillegalArgumentException will be thrown if n or m is less than 1, or if n > m.

Working With Attribute Sets

Merge DICOM Toolkit attribute sets

As mentioned above, the MCattributeSet class encapsulates different types of attribute sets used by DICOM. Sub-classes of MCattributeSet are used to define those different types:

- **MCcommandSet** contains the attributes of a DIMSE message command set.
- **MCdataSet** contains the attributes of a DICOM information object.
- **MCitem** contains the attributes of a DICOM item.
- **MCfileMetaInfo** contains the attributes of a DICOM file's meta information.

Messages and files Contain attribute sets

Your applications deal with network messages in Merge DICOM using MCdimseMessage objects, and DICOM files using MCfile objects. MCdimseMessage objects contain command information attributes (MCcommandSet) and data set attributes (MCdataSet). MCfile objects contain meta information attributes (MCfileMetaInfo) and data set attributes (MCdataSet).

Constructing Message Objects

MCdimseMessage

Network messages are encapsulated in MCdimseMessage objects.

When you call the MCassociation **read** method, Merge DICOM returns a newly constructed and populated MCdimseMessage object.

Constructing a message to use with the MCdimseService base class

When you use the **MCdimseService** class to **send** network messages (as opposed to using one of the MCdimseService sub-classes provided by Merge DICOM Toolkit), you must construct an MCdimseMessage object. You have several options available when you construct the new MCdimseMessage:

Construct a message using a pre-populated data set:

One form of the constructor creates a MCdataSet object and a MCcommandSet object that contain all of the attributes of a DICOM message that will be used for the given *serviceName* and *command*. References to the created attribute sets may be retrieved using the **DataSet** and **CommandSet** properties. Normally you will only deal with the data set and the command set attributes will be set automatically by Merge DICOM Toolkit.

```
MCdimseMessage dm;
uhort command = MCdimseService.C_STORE_RQ;
String serviceName = "STANDARD_CT";
```

```
dm = new MCdimseMessage(command, serviceName);
```

The library uses the *serviceName* parameter to reference the proper message info file along with the data dictionary and builds a MCdataSet object containing all of the attributes that may be used for that the service class.

The *command* parameter is used to inform Merge DICOM what values must be placed in the MCcommandSet it constructs for you.

Exceptions will be thrown if either of the parameters in invalid.

Construct a message with an empty data set:

A second form of the constructor is used if the *service* and *command* are not yet known, or if there is no need to validate the values that will be set. It creates an empty MCdataSet object as well as the MCcommandSet object. The MCdimseMessage object is not associated with any particular DICOM service or command. If the object is to be used to send a message to a network partner, or if the **validate** method is to be called, the **setServiceCommand** method must be called first to associate this message object with a given DICOM service and command.

```
MCdimseMessage dm;

dm = new MCdimseMessage();
```

Construct a message using an existing data set:

A third form of the constructor creates a new MCdimseMessage object that references an existing MCdataSet object for its data set, and constructs a new MCcommandSet object. If the data set has not yet been assigned a service and command, the **setServiceCommand** method should be used to do so before using this object to send a message to a network partner, or if the **validate** method is to be called.

```
MCdimseMessage dm;
MCdataSet ds;  // A non-null reference

dm = new MCdimseMessage(ds);
```

Peformance Tuning

A common programming technique is to construct an empty **MCdataSet** object and use it when setting attribute values (see below).

```
MCdimseMessage dm;
MCdataSet ds = new MCdataSet();  // empty data set

dm = new MCdimseMessage(ds);
```

In this case, the message info and data dictionary files are not accessed when the MCdataSet object is constructed. The MCdataSet object contains no attributes and the **`setServiceCommand`** method must be called to set the service and command for this data set before it can be used in a message sent over the network. Since this approach avoids accessing the message info files, it is more efficient. However, this approach also penalizes you in terms of runtime error checking. This is discussed further later.

Convert an MCfile object

A fourth form of the constructor creates a new MCdimseMessage object that shares the same MCdataSet object as that contained in a specified *file* object.

```
MCdimseMessage dm;
MCfile file;  // A non-null reference

dm = new MCdimseMessage(file);
```

Constructing a message to use with the MCdimseService derived classes

If you will be using one of the sub-classes of the **MCdimseService** class to send network messages, you will not be required to construct a MCdimseMessage object. Typically those DICOM service-class-specific classes require that your provide a data set (MCdataSet) object only.

MCdimseMessage Properties

The MCdimseMessage class contains several methods that can be used to retrieve properties of the DIMSE message.

Transfer Syntax Used

TransferSyntax

The **`TransferSyntax`** property returns an MCtransferSyntax object that identifies the transfer syntax used to encode this message.

Contained attribute sets

MCdimseMessage objects contain references to MCcommandSet and MCdataSet objects. The **`CommandSet`** property returns the MCcommandSet reference and the **`DataSet`** property returns the MCdataSet reference.

The service and command used by the message

Command
ServiceName
setServiceCommand

The **`Command`** property returns the command currently assigned to this message and the **`ServiceName`** property returns the current DICOM service name. The **`setServiceCommand`** method is used to assign a specifiC/C++ DICOM service and command to the message.

MCdimseMessage Command Set Properties

ActionTypeId

Normalized DICOM service classes make use of the N-ACTION DIMSE service. That service requests that a specific action be performed by the peer DIMSE service user. Each SOP class using the N-ACTION service defines Action Type IDs that identify a specific service. The **`ActionTypeId`** property can be used to get or set the Action Type ID that was specified in the DIMSE message. This for attribute (0000,1008).

AffectedSopClassUid

The **`AffectedSopClassUid`** property sets or gets the DICOM Affected SOP Class UID associated with the DIMSE message. It is retrieved from attribute (0000,0002) in the message's command set.

AffectedSopInstanceUid

The **`AffectedSopInstanceUid`** property sets or gets the DICOM "Affected SOP Instance UID" associated with this DIMSE message. It is retrieved from attribute (0000,1000) in the message's command set.

AttributeIdentifiers	DIMSE services using N-GET operations use a command set field to provide an attribute tag for each of the attributes applicable to the N-GET operation. The **`AttributeIdentifiers`** property sets or gets an array of unsigned integer values, each of which is an attribute tag number. This will set or get attribute (0000,1005) in the command set.
CompletedSubOperations	The DIMSE services that use C-GET or C-MOVE operations place the number of C-STORE sub-operations completed in their response messages. The **`CompletedSubOperations`** property gets or sets that value from attribute (0000,1021).
ErrorComment	Many DIMSE services provide for a field to be returned in response messages that describes an error that may occur while servicing a DIMSE request. The **`ErrorComment`** property sets or gets that field from the command set attribute (0000,0902). Usually, the use of this attribute is optional.
ErrorId	Certain DIMSE services provide for sending an application-specific error code in response messages. The **`ErrorId`** property sets or gets that value from the (0000,0903) attribute in the command set.
EventTypeId	Normalized DICOM service classes reference events that are identified by application-specific event IDs. The **`EventTypeId`** property sets or gets the Event Type ID from the command set for attribute (0000,1002).
FailedSubOperations	The DIMSE services that use C-GET or C-MOVE operations place the number of C-STORE sub-operations that failed in their response messages. The **`FailedSubOperations`** property gets or sets that value from attribute (0000,1022).
MessageId	The DIMSE service provider (for example, Merge DICOM Toolkit) assigns a number to each DIMSE request message. The **`MessageId`** property sets or gets that identifying number from the command set attribute (0000,0110).
MessageIdBeingRespondedTo	DIMSE response messages set an attribute in the command set that identifies which request message is being responded to. The **`MessageIdBeingRespondedTo`** property gets that value from attribute (0000,0120).
MessagePriority	The **`MessagePriority`** property gets or sets the DICOM "Message Priority" of this DIMSE message. It is retrieved from attribute (0000,0700) in the message's command set.
MoveDestination	DIMSE C-MOVE request messages contain an attribute that provides the destination DICOM Application Entity for which C-STORE sub-operations are being performed. The **`MoveDestination`** property sets or gets that value from attribute (0000,0600).
MoveOriginator	C-STORE request messages contain an attribute the provides the DICOM AE Title of the DICOM AE which invoked the C-MOVE operation from which a C-STORE sub-operation is being performed. The **`MoveOriginator`** property sets or gets that value from attribute (0000,1030).
MoveOriginatorMessageId	C-STORE request messages contain an attribute that provides the Message ID of the C-MOVE request message from which the C-STORE sub-operation is being performed. The **`MoveOriginatorMessageId`** property gets or sets that value from attribute (0000,1031).

OffendingElements

Some DIMSE services place the data element tag number of the element or elements involved in an error in their response messages. The **`OffendingElements`** property gets or sets these tag numbers, as an array of unsigned integers, from the command set attribute (0000,0901).

RemainingSubOperations

The DIMSE services that use C-GET or C-MOVE operations place the number of C-STORE sub-operations remaining to be sent in their response messages. The **`RemainingSubOperations`** property sets or gets that value from attribute (0000,1020).

RequestedSopClassUid

Normalized DIMSE services provide the SOP Class UID associated with a particular operation in the request and response messages. The **`RequestedSopClassUid`** property sets or gets this "Requested SOP Class UID" from attribute (0000,0003).

RequestedSopInstanceUid

Normalized DIMSE services provide the SOP Instance UID for which a given operation occurred. The **`RequestedSopInstanceUid`** property sets or gets this "Requested SOP Instance UID" from attribute (0000,1001).

ResponseStatus

The **`ResponseStatus`** property gets the Response Status Code (0000,0900) from the message command set.

WarningSubOperations

The DIMSE services that use C-GET or C-MOVE operations place the number of C-STORE sub-operations that generated warnings in their response messages. The **`WarningSubOperations`** property sets or gets that value from attribute (0000,1023).

Constructing File Objects

MCfile

Before you can use the DICOM media storage services provided by the MCmediaStorageService class, you must construct an **MCfile** object that will encapsulate the DICOM file that will be read or written.

As discussed above, each instance of the MCfile object contains an MCdataSet object and an MCfileMetaInfo object. The MCfile may be constructed with a pre-populated data set or with an empty data set.

Construct with a pre-populated data set:

Get the contained attribute sets with getDataSet getMetaInfo

Two forms of the MCfile constructor create a MCdataSet object and a MCfileMetaInfo object that contain all of the attributes of a DICOM file that will be used for the given *`serviceName`* and *`command`*. References to the created attribute sets may be retrieved using the **DataSet** and **MetaInfo** properties. Normally you will only deal with the data set and the file meta information attributes will be set automatically by Merge DICOM Toolkit.

```
MCfile myFile;
ushort command = MCdimseService.C_STORE_RQ;
String serviceName = "STANDARD_CT";

myFile = new MCfile(command, serviceName);

or-

String file = "MyFileName";

myFile = new MCfile( file, command, serviceName );
```

serviceName and *command* are used to access configuration information that describes the attributes of the message. If such configuration information is not available, an empty file object is created, and a warning message is logged. An exception is thrown if the *command* parameter is invalid. The *file* parameter, if used, associates this object with a specific operating system file.

Construct with an empty data set:

Remember:
Service and command are needed to validate the data set

Two forms of the constructor are used if the *service* and *command* are not yet known, or if there is no need to validate that values will be set only for attributes assigned to a given service/command pair. It creates an empty MCdataSet object. The MCfile object is not associated with any particular DICOM service or command. If the validate() method is to be called, the **setServiceCommand** method must be called first to associate this file object with a given DICOM service and command.

```
MCfile myFile = new MCfile();

or-

String file = "MyFileName";
MCfile myFile = new MCfile(file);
```

Convert an MCdimseMessage object to an MCfile object

When you want to save a network message in a DICOM media file

One form of the constructor converts an MCdimseMessage object (*message*) into a file object associated with the specified file system (*file*). The data set contained in *message* will be used in this object.

```
String fileName = "MyFileName";
MCdimseMessage message;  // a non-null reference

myFile = new MCfile(message, fileName);
```

Note: The original MCdimseMessage and the new MCfile objects will be sharing the same MCdataSet object.

Setting data set values

If the command and serviceName parameters are not provided, it is not necessary to add attributes to the data set before setting attribute values. If one of the set value methods of the contained MCdataSet object is used for an attribute, the attribute will automatically be added to the data set before the value is set. This is NOT THE CASE if the MCfile object is built when the command and service are known. In that case the message IS associated with a given service/command pair and attributes other than those associated with that service and command must be explicitly added to the message before setting values for the added attributes.

Specifying the file name

The *fileName* parameter specifies an operating system file that is related to this MCfile object. If the *fileName* parameter is not specified the **File** property defaults to "UNSPECIFIED".

Constructing Item Objects

MCitem objects describe DICOM items used normally in sequence of items (SQ) attributes. They are identified, in Merge DICOM Toolkit, by specific, configured item names.

```
MCitem item = new MCitem();

or-

String itemName = "Configured_Item_Name";
MCitem item = new MCitem(itemName);
```

If the first form of the constructor is used an "empty" MCitem object will be created. The attribute list for the MCitem will initially be empty.

The second constructor form populates the MCitem's attribute list with attributes defined by the *itemName* parameter. The *itemName* is used to access configuration information that describes the attributes of the message. If the *itemName* is unknown to Merge DICOM an empty attribute list is created and a warning message is logged.

Get/Set item name

ItemName

You can use the **ItemName** property of the MCitem class to set or get the item name.

Constructing MCdataSet Objects

DICOM network services (MCdimseService classes) and file service (MCmediaStorageService class) each deal with information objects. These objects are sets of attributes that provide information about the real world entities being acted upon by the service. The MCdataSet class encapsulates such an information object.

The DICOM messages used in DIMSE services (MCdimseMessage class) contain a command set object (MCcommandSet) and an MCdataSet object. The DICOM media storage service deals with file objects (MCfile) that contains a set of file meta information (MCfileMetaInfo class) and an MCdataSet object.

MCdataSet objects are related to specifiC/C++ DICOM service-command pairs (i.e. SOP classes). Normally you construct new instances by providing the service name that identifies the DICOM information object and the DIMSE command that will be used with the information object. By providing the service and command, Merge DICOM can retrieve all of the attributes used for the service and pre-populate the data set with those attributes. (Of course, no values are assigned to the attributes yet – that is your job.)

```
ushort command = MCdimseService.C_STORE_RQ;
String serviceName = "STANDARD_CT";
```

```
MCdataSet ds = new MCdataSet(command, serviceName);
```

The command can be any of the valid constant values defined in the MCdimseService class. (Please refer to the Assembly Windows Help File.) The service name must be one of the services defined in the Services Profile (normally mergecom.srv) file; if not, a warning message will be logged.

An **`MCillegalArumentException`** will be thrown if the command value is invalid. An **`MCconfigurationError`** runtime error will be thrown if the library cannot access the configuration files.

Performance Tuning

A common programming technique is to construct an empty **MCdataSet** object that initially contains no attributes.

```
MCdataSet ds = new MCdataSet();  // empty data set
```

In this case, the message info and data dictionary files are not accessed when the MCdataSet object is constructed. The MCdataSet object contains no attributes and the **`setServiceCommand`** method must be called to set the service and command for this data set before it can be used in a message sent over the network. Since this approach avoids accessing the message info files, it is more efficient. However, this approach also penalizes you in terms of runtime error checking.

Retrieving Contained Attribute Sets

DataSet
CommandSet
MetaInfo

When your application needs to build or parse the attributes contained in MCdimseMessage objects, it accesses the contained MCcommandSet or MCdataSet objects, using the MCattributeSet methods inherited by those classes. Similarly, When your application needs to build or parse the attributes contained in MCfile objects, it accesses the contained MCfileMetaInfo or MCdataSet objects, using the MCattributeSet methods inherited by those classes. Both MCdimseMessage and MCfile classes provide **DataSet** properties to retrieve the contained data set object. The MCdimseMessage provides a **CommandSet** property to retrieve the contained command set and the MCfile class provides a **MetaInfo** property to retrieve the contained file meta info.

MCattribute

Using the MCattribute class

New DICOM attributes can be defined by constructing a new **MCattribute** class object. An MCdataElement object is used to construct the MCattribute.

```
de = new MCdataElement(new MCtag(0x00080010));
MCattribute myAttr = new MCattribute(de);
```

The MCattribute class exists primarily to contain the properties of the DICOM attributes they encapsulate. Properties of the MCattribute class allow you to retrieve the following properties.

- The number of values currently stored for the attribute (the **Count** property).
- The tag that identifies the attribute (the **Tag** property).
- The DICOM Value Representation for the attribute (the **ValueRepresentation** property).

- The values of the attribute.
- The keyword of the attribute.

```
int count = myAttr.Count;
MCtag tag = myAttr.Tag;
MCvr vr = myAttr.ValueRepresentation;
String keyword = myAt.getKeyword();
```

Adding Attributes to an Attribute Set

Adding attributes to the set

Attributes may be added to an attribute set in two ways. The first method is by using the **add** method of the **MCattributeSet** class. The second method is through the use of an indexer. The use of the indexer is defined in the next section.

When using the add method of MCattributeSet, the attribute to be added can be identified by 1) an uint reference or 2) by an MCtag reference. All forms of the add method return the MCattribute added to the set. If the attribute already exists in the set, this attribute is returned.

```
MCattributeSet myAttrSet; // non-null reference
MCtag tag = new MCtag(0x0008, 0x0010);
MCattribute myAttr;

// All of the following accomplish the same thing
myAttr  = myAttrSet.add(tag);
myAttr  = myAttrSet.add(0x00080010);
```

If you are attempting to add a private attribute and there are already 240 private blocks in the attribute's private group, an **`MCinvalidEncodingException`** will be thrown. If the MCtag parameter does not identify an element in the data dictionary or the MCdataElement parameter is not defined in the data dictionary, an **`MCnotStandardElementException`** will be thrown**.**

Using the MCattributeSet indexer to access MCattribute instances

Getting an MCattribute from an attribute set

The MCattributeSet contains a number of indexers for access attributes and values within an attribute set. Two forms of the indexer are specifically for getting and setting MCattribute instances within the MCattributSet. These forms require a uint or MCtag instance to indentify the attribute. The following are the two forms of the indexer setting and getting MCattribute instances:

```
public MCattribute this[uint tag]
public MCattribute this[MCtag tag]
```

The following example shows how the indexers can be used to set and get MCattribute instances from the attribute set.

```
MCtag tag = new MCtag(0x00080010);
MCattribute attrib = new MCattribute(tag);
MCattributeSet myAttrSet; // non-null reference

myAttrSet[tag] = attrib;

attrib = myAttrSet[0x00080010];
```

Removing Attributes from an Attribute Set

Removing a specific attribute from the set

A specific attribute may be removed from an attribute set using the **`removeAttribute`** method of the **MCattributeSet** class. The attribute to be removed can be identified by 1) a uint tag reference or 2) by an MCtag reference. The `removeAttribute` methods do not return a value.

```
MCattributeSet myAttrSet; // non-null reference
MCtag tag = new MCtag(0x0008, 0x0010);

// Both of the following accomplish the same thing
myAttrSet.removeAttribute(tag);
myAttrSet.removeAttribute(0x00080010);
```

The second form of the call will throw an **`MCattributeNotFoundException`** if the attribute is not in the set.

Attribute Properties

Getting/setting attribute properties

The MCattribute class provides three methods to retrieve or set properties of the attribute. The **`ValueRepresentation`** property retrieves the attribute's value representation, the **Count** property returns the number of values assigned to the attribute, and the **`ValueLength`** property returns the length in bytes of this value if it were encoded in a DICOM stream or file.

```
try {
    MCattribute myAttr; // non-null reference
   int count = myAttr.Count;
   MCvr vr = myAttr.ValueRepresentation;
   int length = myAttr.ValueLength;
}  catch (MCattributeNotFoundException e) {…}
   catch (MCincompatibleValueException e) {…}
   catch (MCvrAlreadyValidException e) {…}
```

Assigning Attribute Values from MCattribute

Filling an attribute set with values via MCattribute

Each DICOM attribute may have zero or more values assigned to it, based on the Value Multiplicity assigned to the attribute. You have several methods available to assign values to attributes in an MCattribute object.

- setValue and addValue
- MCattribute[index]
- addEncapsulatedValue

Assigning Attribute Values from MCattributeSet

Filling an attribute set with values via MCattributeSet

The MCattributeSet class also has several convenience methods for assigning values to specific attributes. These routines ensure an attribute has been added into the attribute set, and then set or append the value. There are setValue and addValue routines, similar to the MCattribute class.

Difference between setValue, addValue, and indexer

Do you want to set or add your value?

The "setValue" methods first remove any existing values from the attribute and then append the new value. The "addValue" method appends a value to the attribute.

The indexer can be used to set specific values if the attribute is multi-valued. The following example shows the user of the indexer and the setValue and addValue methods.

```
try {
    MCtag tag = new MCtag(0x000101001);
    MCattribute myAttr = new MCattribute(tag);
    myAttr[0] = new MCpersonName("Smith^John");
    myAttr[1] = new MCpersonName("Smith^Jonathan");

    myAttr.setValue(new MCpersonName("Smith^John"));
    myAttr.addValue(new MCpersonName("Smith^Jonathan"));
}   catch (MCattributeNotFoundException e) {…}
    catch (MCincompatibleValueException e) {…}
    catch (MCvrAlreadyValidException e) {…}
```

Assigning a NULL attribute value

appendNullValue
putNullValue

DICOM allows attributes to have a NULL value (that is the value's length is zero). You can use the `setValue` or addValue routines to assign a value to NULL:

```
MCattributeSet myAttrSet; // non-null reference

myAttrSet.addValue(0x00080010U, null);
myAttrSet.setValue(0x00080020U, null);
```

Note: Both of these methods may be used to set the first or only value of an attribute to NULL (zero-length). The methods may be called to set subsequent values of multi-valued attributes only if the attribute's value representation is a text type that allows the backslash (\) character as a field delimiter in streamed messages. Attributes with the following Value Representations may call this method to set values subsequent to the first value: AE, AS, CS, DA, DS, DT, IS, LO, LT, PN, SH, TM, UI. An **`MCincompatibleVrException`** is thrown if an attempt is made to set a value NULL and the attribute's Value Representation does not allow it.

Assigning a non-NULL attribute

You can use the **`addValue`** or **`setValue`** method to assign non-NULL values. (We will discuss setValue below.)

Identifying the attribute

As discussed above, these methods are implemented in the MCattribute class and the MCattributeSet class, where you must identify the tag that you're working on. This can be done using an MCtag object or by supplying the actual DICOM tag as a uint. The MCattributeSet implementation finds the appropriate MCattribute within the set, and then calls the corresponding setValue or addValue method in the MCattribute class for convenience. All three methods allow you to identify the attribute using an MCtag object.

Polymorphic values

While these methods allow you to specify the value using a variety of data types, the attribute's Value Representation restricts the data type of the value parameter. *Table 11* details which data type may be used with each Value Representation.

Table 11: Permissible data types per Value Representation of the attribute.

Data Type	May be used to set attributes with these VRs
MCdate	DA
MCdateTime	DT
MCtime	TM
MCage	AS
String	AE, DS, IS, UI, CS, LO, LT, SH, ST, UT, DA, DT, TM, AS, PN, FL, FD, AT, UL, SL, SS, US
MCpersonName	PN
float	FL, FD, UL, AT, DS, IS
double	FD, FL, UL, AT, DS, IS
uint	SL, SS, US, UL, AT, IS
int	SL, SS, US, UL, AT, IS
short	SL, SS, US, UL, AT, IS
ushort	SL, SS, US, UL, AT, IS
byte[ ]	OB, OF, OW, UNKNOWN_VR
MCdataSource	OB, OF, OW, UNKNOWN_VR,
MCitem	SQ

```
MCattributeSet myAttrSet; // non-null reference
MCtag tag1 = new MCtag(0x00080020);
MCtag tag2 = new MCtag(0x00080030);
MCtag tag3 = new MCtag(0x00080052);
MCtag tag4 = new MCtag(0x00280010);
// the value can be any of the types shown in Table 11.
String value = "my value";

myAttrSet.setValue(tag1, new MCdate("20051109"));
myAttrSet.setValue(tag2, new MCtime("081401");
myAttrSet.putValue(tag3, "PATIENT");
myAttrSet.setValue(tag4, (ushort)256);
```

Values are converted to the proper Value Representation

Merge DICOM will perform any reasonable conversion from the types listed in . *Table 11* to the form necessary to encode it in the Value Representation of the attribute. If a type conversion is not reasonable (e.g., from `short` to LT), then an **`MCincompatibleValueException`** will be thrown. An **`MCinvalidEncodingWarning`** will be thrown if the value is invalid according to the rules for the value representation. Note that this is just a warning – the value is encoded.

Note that each time the addValue method is called for an attribute, another value is added to the attribute's list of values.

Using alternate character sets

A default string encoder is implemented that will convert between Unicode and many of the DICOM defined character sets. If you want to define a string encoder or decoder that is different than the default implementation, you must use the **`MC.mcSetStringEncoder`** method to set your own string encoder and decoder.

To set values for attributes with value representations of OB, OW, OF use the `setValue` call which includes **MCdataSource** as a parameter. The MCdataSource class is described further in the following section. The `setValue` may also be used to set values for attributes of types SL, SS, UL, US, AT, FL and FD.

Supplying pixel data

MCdataSource provideData method
MCdata class
MCcallbackCannotComplyException

setValue

Using an MCdataSource class to assign an attribute value

When setting the value of an attribute with a value representation of OB or OW or OF (e.g., Pixel Data), you can create a class that implements the **MCdataSource** interface and then call **setValue(MCdataSource, uint)** method of MCattribute to assign the attribute's value. Pixel Data can be very large and you can use this method to supply the data value a 'chunk' or block at a time.

The callback class you provide must provide a **provideData** method that is called by the library to retrieve portions of the attribute's value. The library provides the provideData method a bool that is true the first time the method is called to retrieve attribute values. The library also provides a System.Object reference to the instance that is calling provideData. The provideData method is required to return portions of the attribute's value, using an instance of the **MCdata** class.

As an example, your application could define a MCdataSource class called `MyPDSupplyCallback` whose purpose is to supply Pixel Data. Pseudo-code for this class follows:

```
class MyPDSupplyCallback : MCdataSource {
   public String file = null;
   public MCdata provideData(bool isFirst Object origin)  {
      if (isFirst) {
         // Open pixel data source (e.g.,  a file) here
         …
         if (openFailed)
            throw new MCcallbackCannotComplyException();
         }

      // Read next chunk of pixel data from source
      // and return it and its size in a MCdata object
         …
      if (readFailed)
         throw new MCcallbackCannotComplyException();
```

```
            // Data is read into chunk
            byte[] chunk = new byte[4096];
            // put number of bytes read in size
            int size;
            // set isLast to true if this is last of the data
            bool isLast;

            MCdata data = new MCdata(chunk, size);
            data.IsLast = isLast;
            return data;
        }
    }
```

This MCdataSource class is called by Merge DICOM only when triggered by your application. For example, your application might use MyPDSupplyCallback to set the value of the `MCdicom.PIXEL_DATA` attribute `(7FE0, 0010)` as follows:

```
MCattributeSet as;  // non-null reference
uint length;
MyPDSupplyCallback cb = new MyPDSupplyCallback();
cb.file = "MypixelDataFile";

MCattribute attrib = as[MCdicom.PIXEL_DATA];
attrib.setValue(cb, length);
```

On making this call, the toolkit library will keep a reference to the MCdataSource instance. When the data is required by the toolkit (if the attribute set is written to a file or to the network), it will repetitively call the provideData method of the callback class until it indicates that all the pixel data has been read in without any errors. If your callback class throws `MCcallbackCannotComplyException`, the library will fail its current operation.

Performance Tuning

Supplying Pixel Data a block at a time is especially useful for very large Pixel Data and/or on platforms with resource (e.g., memory) limitations. In this case you would also want to set `LARGE_DATA_STORE` to the value `FILE` in the Service Profile, and Merge DICOM Toolkit will store the Pixel Data value in a temporary file.

If your application runs on a resource rich system, you should set `LARGE_DATA_STORE` to the value `MEM` in the Service Profile, and Merge DICOM Toolkit will keep the Pixel Data values in the message object stored in memory rather than using temporary files. This should improve performance. Also, in this case you may want your callback class to supply the Pixel Data in fewer big blocks (or one large block).

MCfileDataSource
MCstreamDataSource
MCmemoryDataSink

Merge DICOM provides a number of implementations to the MCdataSource interface. The **MCfileDataSource** class implements a data source that reads from a file. A file name is supplied to the constructor, and the class will automatically supply the contents of this file. The **MCstreamDataSource** class implements a data source that reads from a `System.IO.Stream` derived instance. For example, an instance of the **System.IO.FileStream** class can be used to read the file. Finally, the **MCmemoryDataSink** class also implements the MCdataSource interface. This class also implements the MCdataSink interface, which is described in a subsequent section. The MCmemoryDataSink class implements a data source where the data is supplied from memory. The MCmemoryDataSink class constructor takes an MCdata instance in the constructor which contains the actual data being supplied.

Retrieving Attribute Values

When your AE receives a DICOM message, it will most often need to examine the values contained in the message attributes to perform an action (e.g., store an image, print a film, change state...). If your application is a server, the message conveys the operation your server should perform and the data associated with the operation. If your application is a client, the message may be a response message from a server on the network resulting from a previous request message to that same server.

Retrieving values from an attribute set

Indexers

Once you have received an MCattributeSet object (probably one contained in an MCdimseMessage object), you can use the indexer for MCattributeSet to retrieve values. (Note that you can also use a different form of the indexer to retrieve MCattribute objects, which in turn have an indexer to retrieve values.) The indexers have the following forms:

```
public Object this[uint tag, int index];
public Object this [MCtag tag, int index];
public Object this [uint tag, int index, Object defaultValue];
public Object this [MCtag tag, int index, Object defaultValue];
```

Each of these methods return an Object representing the value. The tag to retrieve can be specified as an MCtag instance or a uint containing the tag. The *index* parameter allows the user to specify the specific value to get, if the attribute has a value of multiplicity greater than one. The index is zero based. Two final forms are added as a convenience and allow a default value to be specified if the attribute does not have a value or it is missing in the attribute set.

Each method returns an Object representing the value; the type of object returned is determined by value representation of the attribute that is being retrieved. The methods return `null` if the attribute's value was a DICOM NULL value (i.e. its value length was zero). *Table 12* below shows the data types that are returned for each DICOM Value Representation.

Table 12: Valid value type parameters for the various Value Representations

Data Types Returned	Value Representations
MCdate	DA
MCdateTime	DT
MCtime	TM
MCage	AS
String	AE, DS, IS, UI, CS, LO, LT, SH, ST, UT
MCpersonName	PN
float	FL
double	FD
uint	UL, AT

Data Types Returned	Value Representations
int	SL
short	SS
ushort	US
MCdataSink	OB, OF, OW, UNKNOWN_VR,
MCitem	SQ

Note: If a value is retrieved which is not set, the indexer will throw an MCnoSuchValueException.

```
MCdimseMessage dm;   // non-null reference
MCattributeSet as = dm.getDataSet();
try {
    MCdate value = (MCdate)as[MCdicom.
        INSTANCE_CREATION_DATE, 0);
} catch (MCnoAttributesException e) {…}
  catch (MCattributeNotFoundException e) {…}
  catch (MCincompatibleValueException e) {…}
  catch (MCnoSuchValueException e) {…}
```

The MCattribute class contains a number of routines that can do explicit conversion from the internal encoding for VRs into other data types. These MCattribute methods are **`getIntValue`**, **`getStringValue`**, **`getUIntValue`**, and **`getDoubleValue`**. These routines will convert the internal Merge DICOM representation into the types specified in the routine name. See the Assembly Windows Help File for further details on these methods.

Using a Callback Class to Retrieve an Attribute's Value

Have Merge DICOM Toolkit give a callback the value

MCdataSink receiveData method

You shall use the **IsBulk** attribute's property to identify if the value is bulk. To retrive the bulk value you must use the **`readBulkData`** method or the **`readNextFrame`** method of MCattribute.. In most of the cases it is a value representation of OB or OW or OF (e.g., Pixel Data). Pixel Data tends to be very large and normally you use this method to read the data value a 'chunk' or block at a time. This method is the complement to the `setValue` method described previously.

You must construct a callback class that implements the **MCdataSink** interface and then call `readBulkData` to retrieve the attribute's value.

The callback class you provide must provide a **`receiveData`** method that is called by the library to provide portions of the attribute's value. The library calls the `receiveData` method, passing an instance of the MCdata class that contains a reference to the data being provided and a System.Object reference to the origin of the data.

As an example, your application could define a MCdataSink class called `MyPDStoreCallback` whose purpose is to store Pixel Data to an external data sink so that your application uses less primary memory. Pseudo-code for this class follows:

```
class MyPDStoreCallback : MCdataSink {
    public String file = null;
    public bool isFirst = true;
    public void receiveData (MCdata data,
        System.Object origin) {
        if (isFirst) {
            isFirst = false;
            // Open pixel data sink (e.g., file) here
            …
            if (openFailed)
                throw new MCcallbackCannotComplyException();
        }

        byte[] array = data.ManagedBuffer;
        int size = data.Length;

        // Store size bytes of the array in the pixel data sink.
        …
        if (storeFailed)
            throw new MCcallbackCannotComplyException();
        if (data.IsLast) {
        // close the data sink here
        }
        return;
    }
}
```

This callback is called by the Merge DICOM .NET Library only when triggered by your application. For example, your application might use `MyPDStoreCallback` to retrieve the value of the `TAG_PIXEL_DATA` attribute `(7FE0,0010)` as follows:

```
MCdataSet ds;  // non-null reference
MyPDStoreCallback cb = new MyPDStoreCallback();
cb.file = "MypixelDataFile";
try {
    MCattribute attrib = ds[MCdicom.PIXEL_DATA];
    attrib.readBulkData(cb);
} catch (MCexception e) {…}
```

On making this call, the toolkit library will repetitively call the receiveData method of the `MyPDStoreCallback` class until all the pixel data has been retrieved from the attribute without any errors.

Performance Tuning

Storing or 'setting aside' Pixel Data a block at a time is especially useful for very large Pixel Data and/or on platforms with resource (e.g., memory) limitations. In this case, you would also want to set `LARGE_DATA_STORE` to the value `FILE` in the Service Profile, so that Merge DICOM Toolkit will also maintain the pixel data value stored in the attribute set in a temporary file.

If your application runs on a resource rich system, you should set `LARGE_DATA_STORE` to the value `MEM` in the Service Profile, and Merge DICOM Toolkit will keep the pixel data values in the attribute set stored in memory rather than using temporary files. This should improve performance. Also, in this case you may want your callback class to store the Pixel Data in fewer big blocks (or one large block) and keep them in primary memory for rapid access.

MCfileDataSink
MCstreamDataSink
MCmemoryDataSink

Merge DICOM provides a number of implementations to the MCdataSink interface. The **MCfileDataSink** class implements a data sink that writes to a file. A file name is supplied to the constructor, and the class will automatically write the supplied data to this file. The **MCstreamDataSink** class implements a data sink that writes to a `System.IO.Stream` derived instance. For example, an instance of the **System.IO.FileStream** class can be used to write to a file. Finally, the **MCmemoryDataSink** class also implements the MCdataSink interface. This class implements a data sink where the data is stored in memory.

Retrieving an Attribute Value's Properties

Get a value's length

MCattribute.ValueLength

You can obtain the length of an attribute's value by using the **`ValueLength`** property of the MCattribute class. The length returned is the stream length of the attribute. It is the sum of all lengths of all values if the attribute is multi-valued. If the VR is a text VR and the attribute is multi-valued, the length also includes the numbers of separators.

If the attribute's value is a DICOM NULL, zero is returned.

If an attribute has a value representation of SQ, the number of items in the sequence is returned.

```
MCattributeSet as;  // non-null reference
MCtag tag = new MCtag(0x0008, 0x0010);

uint length = as[tag].ValueLength;
```

Please refer to the Assembly Windows Help File for the exceptions that may be thrown.

Get a count of values

MCattribute.Count

You can use the **Count** property of the MCattribute class to retrieve the number of values that are currently stored for an attribute. If no values are stored for the attribute zero will be returned. If the attribute contains one NULL value, 1 will be returned.

```
MCattributeSet as;  // non-null reference
MCtag tag = new MCtag(0x0008, 0x0010);

int values = as[tag].Count;
```

Listing an Attribute Set

Attribute set report

list method

You can create a formatted list of the attributes of an attribute set, along with their values by using the **`list`** method of the MCattributeSet. The `list` method produces a report describing the contents of the MCattributeSet (or one of its sub-classes). The report will be written to the `TextWriter` provided, or to stdout, if a file is not provided.

Note: If the object contains an attribute with a Value Representation of SQ (sequence of items), each item in the sequence will be listed. Each sequence of items is indented in the listing four spaces to the right of its owning message or items.

```
MCattributeSet as;  // a non-null reference
as.list();    // list to the standard output stream
System.IO.StreamWriter writer = new StreamWriter("myFile");
as.list(writer);  // list to myFile
```

Converting an Attribute Set into an XML string

Attribute set to XML conversion

writeToXML method

You can convert a list of attributes of an attribute set, along with their values into XML string by using the **writeToXML** method of the MCattributeSet. The writeToXML method creates an XML string describing the contents of the MCattributeSet. The XML buffer is written to the stream identified by the stream object provided.

Note: If the object contains an attribute with a Value Representation of SQ (sequence of items), each item in the sequence will be converted into its XML representation.

The following example shows how the writeToXML method is utilized at a high level.

```
MCdataSet ds; // a non-null reference
MCxmlOptions xmlOptions = MCxmlOptions.XmlOptIncludeBulks |
      XmlOptExcludeSequences;
StreamWriter writer = new StreamWriter( "myXMLFile" );
// convert DICOM DataSet to an XML file
ds.writeToXML( writer, xmlOptions );
writer.Close();
```

The following configuration flags are defined in the MCxmlOptions enumeration and are available for the Attribute Set to XML conversion.

```
// Use the default settings
XmlOptDefault                  = 0x0
// Store bulk attributes (VR is OB or OW) in the XML
XmlOptIncludeBulks             = 0x1
// Store Pixel Data buffer in the XML
XmlOptIncludePixelData         = 0x2
// Do not store Sequence attributes in the XML
XmlOptExcludeSequences         = 0x4
// Do not store Private attributes in the XML
XmlOptExcludePrivateAttributes = 0x8
// Use Base64 encoding for bulks and UN VR attributes
XmlOptBase64Binary             = 0x10
```

Converting an XML string into an Attribute Set

XML to Attribute set conversion

readFromXML

You can read attribute values from an XML string into an attribute set by using the **readFromXML** method of the MCattributeSet.

The content of the attribute set is not cleared before processing XML attributes. The existing attributes in the attribute set will be overridden if they are present in the XML string.

The following example shows how the readFromXML method is utilized at a high level.

```
StreamReader reader = new StreamReader( "myXMLFile" );
MCdataSet ds = new MCdataSet();
// convert an XML file into an attribute set
ds.readFromXML( reader );
reader.Close();
```

8-bit Pixel Data

Swapping pixel data bytes

Swap Method

For DICOM's Implicit VR Little Endian transfer syntax, the pixel data attribute's `(7fe0,0010)` VR is specified as being OW (independent of what the bits allocated and bits stored attributes are set to). To reduce confusion, Merge DICOM Toolkit sets the VR of pixel data for the other non-encapsulated transfer syntaxes to OW.

When retrieving or setting pixel data with a `MCdataSink or MCdataSource` class, the toolkit assumes that the OW pixel data is encoded in the host system's native endian format as defined by DICOM. *Figure 12* describes how 8-bit pixel data is encoded in an OW buffer for both big and little endian formats.

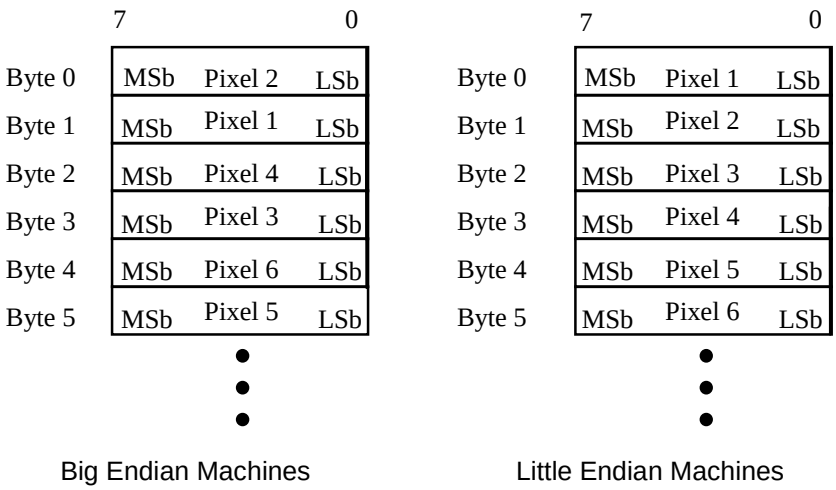

Figure 12: Sample Pixel Data Byte Stream for 8-bits Allocated, 8-bits Stored, High bit of 7 (VR = OW)

The DICOM standard specifies that the first pixel byte should be set to the *least significant byte* of the OW value. The next pixel byte should be set to *the most significant byte* of the OW value. This implies that on big endian machines, 8-bit pixel data is byte-swapped from the OB encoding method. Note that .NET and Windows is in Little Endian format.

Encapsulated Pixel Data

Handling Compressed Data

Merge DICOM Toolkit supports the DICOM encapsulated transfer syntaxes. The method for compression and decompression (JPEG, RLE, etc.) of encapsulated pixel data is specified in part 5 of the DICOM standard. Merge DICOM also supports compression and decompression for several specific transfer syntaxes. The methods for this are discussed in subsequent section. Besides this support, the MCattribute class also has several methods for dealing with encapsulated transfer syntaxes.

There are several classes that can be used for compressing or decompressing data. See "Compression and decompression" section for further details.

Encapsulated pixel data is dealt with in a similar manner as standard pixel data. MCdataSink and MCdataSource classes are used. Merge DICOM .NET has the capability of encapsulating each frame of data according to the encoding specified in Part 5 of the DICOM standard. *Table 13* contains a sample encoding of frames.

Table 13: Sample Encapsulated Pixel Data

Pixel Data Element									
Basic Offset Table with NO Item Value		First Fragment (Single Frame) of Pixel Data			Second Fragment (Single Frame) of Pixel Data			Sequence Delimiter Item	
Item Tag	Item Length	Item Tag	Item Length	Item Value	Item Tag	Item Length	Item Value	Sequence Delim. Tag	Item Length
(FFFE, E000)	0000 0000H	(FFFE, E000)	0000 04C6H	Compressed Fragment	(FFFE, E000)	0000 024AH	Compressed Fragment	(FFFE, E0DD)	0000 0000H
4 bytes	4 bytes	4 bytes	4 bytes	04C6H bytes	4 bytes	4 bytes	024A H bytes	4 bytes	4 bytes

When encoding encapsulated data, each compressed fragment shown in *Table 13* is set with the **`addEncapsulatedValue`** method of MCattribute. This routine supplies an MCdataSource which contains the encapsulated pixel data. When needed, Merge DICOM will retrieve data from the source and encapsulate. In this case, it is assumed that the MCdataSource is supplying already compressed data. `addEncapsulatedValue` can be called multiple times to add additional compressed frames to an attribute.

When decoding encapsulated data, the **`readNextFrame`** routine of MCattribute is used. This method can be repeatedly called to retrieve each encapsulated frame. Note that the compressed data is returned.

Working with MCabstractMessage Derived Classes

Abstract Message Routines

MCabstractMessage

The **MCabstractMessage** class is an abstract class that implements several routines that are common between DIMSE messages (MCdimseMessage) and DICOM files (MCfile). These routines include compression and validation of attribute sets.

Compression and Decompression

The MCabstractMessage derived classes (MCfile and MCdimseMessage) provide a **duplicate** method, which can be utilized to do compression, decompression, or both. The duplicate method will create a copy of the MCfile or MCdimseMessage that is encoded in a new transfer syntax. If the transfer syntax of the source message is compressed, a decompressor must be supplied to the duplicate method. If the result transfer syntax is compressed, a compressor must be supplied.

MCcompression

The **MCcompression** interface defines an interface for compressors and decompressors utilized by Merge DICOM .NET. Merge DICOM supplies a number of compressors and decompressors that implement this interface and are defined in the following sections. The duplicate method of MCabstractMessage requires that the compressor and decompressor supplied to it implement the MCcompression interface. The following example shows how the duplicate method is utilized at a high level.

```
MCdimseMessage msg; // A non-null uncompressed message

MCdimseMessage resultMsg;

resultMsg = msg.duplicate( MCtransferSyntax.Rle, null,
      new MCrleCompressor());

MCdimseMessage resultMsg2;

resultMsg2 = resultMsg.duplicate(
MCtransferSyntax.JpegBaseline,
new MCrleDecompressor(),
new MCstandardCompressor());
```

The example starts with an uncompressed image, and creates a new message that is RLE compressed with the duplicate method. It then calls duplicate again to decompress the RLE image and recompress the image as JPEG Baseline.

The following section describes the compressors and decompressor supplied by Merge DICOM and how they must be utilized with MCabstractMessage.duplicate.

Merge DICOM Supplied Compressors and Decompressors

MCstandardCompressor
MCstandardDecompressor
MCrleCompressor
MCrleDecompressor

Merge DICOM supplies several implementations of compressors and decompressors. *Table 14* lists each of the Merge DICOM classes, if they implement compression or decompression and what transfer syntaxes they support.

Table 14: Merge DICOM Supplied Compressor and Decompressors

Merge DICOM Class	Type	DICOM Transfer Syntaxes Supported
MCstandardCompressor	Compressor	JPEG Baseline JPEG Extended (Process 2 & 4) JPEG Lossless Non-Hierarchical Process 14 JPEG 2000 JPEG 2000 Lossless Only
MCrleCompressor	Compressor	RLE
MCstandardDecompressor	Decompressor	JPEG Baseline JPEG Extended (Process 2 & 4) JPEG Lossless Non-Hierarchical Process 14 JPEG 2000 JPEG 2000 Lossless Only
MCrleDecompressor	Decompressor	RLE

For the JPEG Baseline, Jpeg Extended (Process 2 & 4), JPEG Lossless Non-Hierarchical Process 14, JPEG 2000, and JPEG 2000 Lossless Only transfer syntaxes, Merge DICOM utilizes libraries from Pegasus Imaging to do compression and decompression. The RLE transfer syntax is supported directly in Merge DICOM Toolkit.

Pegasus Licenses are required to remove speed limitations

JPEG Baseline, JPEG Extended (Process 2 & 4), and JPEG Lossless Non-Hierarchical Process 14 can be compressed or decompressed at a maximum rate of 3 images (or frames) per second. For JPEG 2000 Lossless and Lossy, a dialog will be displayed each time the compressor or decompressor is used. Full licenses can be purchased from Pegasus and configured in Merge DICOM to remove these compression and decompression limits. The licenses can be configured in the mergecom.pro configuration file.

The MCtransferSyntax.JpegBaseline transfer syntax is UID 1.2.840.10008.1.2.4.50, JPEG Baseline (Process 1): Default Transfer Syntax for Lossy JPEG 8 Bit Image Compression, and uses Pegasus libraries 6420/6520. *Table 15* details the photometric interpretation and bit depths supported by the standard compressor and decompressor for this transfer syntax. When lossy compressing RGB data, the standard compressor by default compresses the data into `YBR_FULL_422` format. The compressor can also compress in `YBR_FULL` format if the `COMPRESSION_RGB_TRANSFORM_FORMAT` configuration option is set to `YBR_FULL`. The Photometric Interpretation tag must be changed by the application after compressing RGB data. Similarly, the Photometric Interpretation tag should be changed back to RGB before decompressing `YBR_FULL` or `YBR_FULL_422` data.

Table 15: JPEG Baseline Supported Photometric Interpretations and Bit Depths

JPEG Baseline			
Photometric Interpretation	MONOCHROME1 MONOCHROME2	RGB	YBR_FULL_422 YBR_FULL
Bits Stored	8	8	8
Bits Allocated	8	8	8
Samples Per Pixel	1	3	3

The MCtransferSyntax.JpegExtended2_4 transfer syntax is UID 1.2.840.10008.1.2.4.51, JPEG Extended (Process 2 & 4): Default Transfer Syntax for Lossy JPEG 12 Bit Image Compression (Process 4 only), and uses Pegasus libraries 6420/6520. *Table 16* details the photometric interpretation and bit depths supported by the standard compressor and decompressor for this transfer syntax. When lossy compressing `RGB` data, the standard compressor by default compresses the data into `YBR_FULL_422` format. The compressor can also compress in `YBR_FULL` format if the `COMPRESSION_RGB_TRANSFORM_FORMAT` configuration option is set to `YBR_FULL`. The Photometric Interpretation tag must be changed by the application after compressing `RGB` data. Similarly, the Photometric Interpretation tag should be changed back to `RGB` before decompressing `YBR_FULL` or `YBR_FULL_422` data.

Table 16: JPEG Extended Supported Photometric Interpretations and Bit Depths

JPEG Extended (Process 2 & 4)					
Photometric Interpretation	MONOCHROME1 MONOCHROME2			RGB	YBR_FULL_422 YBR_FULL
Bits Stored	8	10	12	8	8
Bits Allocated	8	16	16	8	8
Samples Per Pixel	1	1	1	3	2

The MCtransferSyntax.JpegLosslesHier14 transfer syntax is UID 1.2.840.10008.1.2.4.70, JPEG Lossless, Non-Hierarchical, First-Order Prediction (Process 14 [Selection Value 1]): Default Transfer Syntax for Lossless JPEG Image Compression, and uses Pegasus libraries 6220/6320. *Table 17* details the photometric interpretation and bit depths supported by the standard compressor and decompressor for this transfer syntax. The standard compressor does not do a color transformation to RGB data when compressing with `JPEG_LOSSLESS_HIER_14`. The Photometric Interpretation tag should be left as `RGB` in this case.

Table 17: JPEG Lossless Supported Photometric Interpretations and Bit Depths

JPEG Lossless Non-Hierarchical Process 14		
Photometric Interpretation	MONOCHROME1 MONOCHROME2	RGB
Bits Stored	2 to 16	8
Bits Allocated	8 or 16	8
Samples Per Pixel	1	3

The MCtransferSyntax.Jpeg2000 transfer syntax is UID 1.2.840.10008.1.2.4.91, JPEG 2000 Image Compression, and uses Pegasus libraries 6820/6920 for lossy or lossless. *Table 18* details the photometric interpretation and bit depths supported by the standard compressor and decompressor for this transfer syntax.

Table 18: JPEG 2000 Lossy Supported Photometric Interpretations and Bit Depths

JPEG 2000 (When used for Lossy)						
Photometric Interpretation	MONOCHROME1 MONOCHROME2				YBR_ICT	RGB
Bits Stored	8	10	12	16	8	8
Bits Allocated	8	16	16	16	8	8
Samples per Pixel	1	1	1	1	3	3

The MCtransferSyntax.Jpeg2000LosslessOnly transfer syntax is UID 1.2.840.10008.1.2.4.90, JPEG 2000 Image Compression (Lossless Only), and uses Pegasus libraries 6820/6920 for lossless. *Table 19* details the photometric interpretation and bit depths supported by the standard compressor and decompressor for this transfer syntax.

Table 19: JPEG 2000 Lossless Supported Photometric Interpretations and Bit Depths

JPEG 2000 Lossless						
Photometric Interpretation	MONOCHROME1 MONOCHROME2				YBR_RCT	RGB
Bits Stored	8	10	12	16	8	8
Bits Allocated	8	16	16	16	8	8
Samples Per Pixel	1	1	1	1	3	3

SPECIAL NOTES!

When using the standard compressor, all data needs to be right justified, i.e. bit 0 contains data, but the highest bits may not. RGB and YBR must be non-planar (R1G1B1, R2G2B2, ... or Y1Y2B1R1, Y3Y4B3R3,...)

MCtransferSyntax.Jpeg2000 and MCtransferSyntax.Jpeg2000LosslessOnly will cause a irreversible, or reversible color transformation when compressing RGB data. The Photometric Interpretation MUST be changed from `RGB` to:

- `YBR_ICT` if MCtransferSyntax.Jpeg2000 is used with `COMPRESSION_WHEN_J2K_USE_LOSSY = Yes` (Lossy color transform for lossy compression)
- `YBR_RCT` if MCtransferSyntax.Jpeg2000LosslessOnly or MCtransferSyntax.Jpeg2000 with `COMPRESSION_WHEN_J2K_USE_LOSSY = No` (Lossless color transform for lossless compression).

Similarly, on the decompression end, the Photometric Interpretation should be changed back to `RGB`, but the Lossy Image Compression attribute should indicate it has been lossy compressed.

Validating Attribute Sets

You can validate MCdimseMessage, MCfile, and MCdataSet instances

Validate method

Once your application has a populated message object, either one that you have built or one that you have received and are about to parse, Merge DICOM Toolkit supplies DICOM Toolkit DICOM message validation functionality. The MCabstractMessage derived classes (MCdimseMessage and MCfile) and the MCdataSet class each provide a **`validate`** method that will validate the attribute sets it contains against the DICOM Standard's specification for its service-command pair.

`message.txt` can be very useful

One of the files supplied with Merge DICOM Toolkit is the `message.txt` file. This file contains a listing of all the messages supported by the toolkit and the parameters they are validated against. `message.txt` is a useful guide in your application development because it specifies the attributes that can make up the object instance portion of each message type (service-command pair) and is often easier to use as a quick reference than paging through two or three parts of the DICOM Standard. `message.txt` also specifies the contents of items and files (see discussions of Sequence of Items and DICOM Files later in this document). Remember though that the DICOM Standard is the final word and that `message.txt` has its limitations as described further below.

The `validate` methods do not validate the attributes that make up the command portion of a DICOM message. Command set attributes (attributes with a group number less than 0008) are also not specified in `message.txt`. The Merge DICOM Toolkit Library sets as many of the command group attributes as possible automatically. In some services, your application may need to set command set attributes if you do not use one of the sub-classes of the MCdimseService class.

An excerpt of `message.txt` follows for the service-command pair `DETACHED_PATIENT_MANAGEMENT - N_GET_RSP` as an illustration. For each attribute in the message, at least one line of data is specified. This first line includes the tag, attribute name, value representation, and value type. Additional lines may be included for the attribute to list conditions, enumerated values, defined terms, and item names for attributes with a VR of SQ. You should refer to the DICOM Standard (parts 3 and 4) for a detailed description of particular conditions and their meanings.

DETACHED_PATIENT_MANAGEMENT - N_GET_RSP

```
0008,0005    Specific Character set                    CS     1C
Condition: EXPANDED_OR_REPLACEMENT_CHARACTER_SET_USED
Defined Terms: ISO-IR 100, ISO-IR 101, ISO-IR 109, ISO-IR 110,
ISO-IR144, ISO-IR 127, ISO-IR 126, ISO-IR 138, ISO-IR 148
0008,1110    Referenced Study Sequence                 SQ     2
Item Name(s): REF_STUDY
0008,1125    Referenced Visit Sequence                 SQ     2
Item Name(s):
0010,0010    Patient's Name                            PN     2
0010,0020    Patient IDLO2
0010,0021    Issuer of Patient ID                      LO     3
0010,0030    Patient's Birth Date                      DA     2
0010,0032    Patient's Birth Time                      TM     3
0010,0040    Patient's Sex                             CS     2
Enumerated Values: M, F, O
0010,0050    Patient's Insurance Plan Code Sequence    SQ     3
Item Name(s): PATIENTS_INSURANCE_PLAN_CODE
0010,1000    Other Patient IDs                         LO     3
0010,1001    Other Patient Names                       PN     3
0010,1005    Patient's Birth Name                      PN     3
0010,1020    Patient's Size                            DS     3
0010,1040    Patient's Address                         LO     3
0010,1060    Patient's Mother's Birth Name             PN     3
0010,1080    Military Rank                             LO     3
0010,1081    Branch of Service                         LO     3
0010,1090    Medical Record Locator                    LO     3
0010,2000    Medical Alerts                            LO     3
0010,2110    Contrast Allergies                        LO     3
0010,2150    Country of Residence                      LO     3
0010,2152    Region of Residence                       LO     3
0010,2154    Patient's Telephone Numbers               SH     3
0010,2160    Ethnic Group                              SH     3
0010,21A0    Smoking Status                            CS     3
Enumerated Values: YES, NO, UNKNOWN
0010,21B0    Additional Patient History                LT     3
0010,21C0    Pregnancy Status                          US     3
Enumerated Values: 0001, 0002, 0003, 0004
0010,21D0    Last Menstrual Date                       DA     3
0010,21F0    Patient's Religious Preference            LO     3
0010,4000    Patient Comments                          LT     3
0038,0004    Referenced Patient Alias Sequence         SQ     2
Item Name(s): REF_PATIENT_ALIAS
0038,0050    Special Needs                             LO     3
0038,0500    Patient State                             LO     3
```

What validation can do for you...

While Merge DICOM validation is not foolproof, it is very useful and will catch many standard violations. It validates the following:

- That the value assigned to an attribute is appropriate for that attributes VR.
- That all value type 1 attributes have a value, and that value is not null.
- That all value type 2 attributes have a value, and that value may be null.
- That a specified set of conditional attributes (value type 1C or 2C) are validated as value type 1 or 2 attributes when the specified condition is satisfied. Merge DICOM supports a number of conditional functions that are straightforward to validate. Not all conditions can be validated by the toolkit and those that cannot need to be checked by the application itself.
- That an attribute does not have too many or too few values for its specified value multiplicity.
- That an attribute that has enumerated values does not have a value that is not one of the enumerated values. A warning is also issued if an attribute that has defined terms has a value that is not one of those defined terms.
- That a non-private attribute is not included in the message that is not defined for that DICOM message (service-command pair).

and what validation cannot do for you

As mentioned, Merge DICOM Toolkit does not capture all standard violations, and the DICOM Standard itself should be considered the final word when validating a message. Important limitations of Merge DICOM validation include:

- DICOM Part 3 specifies Information Object Definitions (IOD's) as being composed of modules. Each module contains attributes. Only in the case of composite IOD's may an attribute be specified in DICOM Part 3 as being contained in either a User Optional or Conditional Module. Merge DICOM Toolkit treats all such attributes as being value type 3 (optional).
- Also, certain modules may be mutually exclusive (e.g., curve and overlay modules), in the case of some composite IOD's (e.g., Ultrasound Image Object) used in storage services.
- For normalized services using the N-EVENT-REPORT command, the actual contents of an N-EVENT-REPORT message are dependent on the Event Type ID being communicated. Merge DICOM Toolkit treats all Event Type ID's identically when performing message validation; namely it treats all attributes as type 3.

An example...

An example of the use of the validate method follows. The example assumes a MCdimseMessage (`msg`) was just received and the intent is to validate the message.

```
MCdimseMessage msg;  // non-null reference

bool validates;
validates = msg.validate(MCvalidationLevel.Errors_Only);
if (!validates)
{
    MCvalidationError err = msg.getNextValidationError();
```

```
        while (err != null)
        {
            System.Console.Out.WriteLine(err.ToString());
            err = msg.getNextValidationError();
        }
    }
```

MCvalidationLevel enum

MCvalidationError firstError nextError

In this example, the application validates the MCdimseMessage object `msg` at the `MCvalidationLevel.Errors_Only` level. `MCvalidationLevel.Errors_And_Warnings` could be used to report both warnings and errors, while `MCvalidationLevel.Full` could be used to report errors, warnings, and informational messages. If MCdimseMessage.validate returns false, your application can use the **`getNextValidationError`** method to retrieve **MCvalidationError** objects that describe the error. Each MCvalidationError instance has these public properties:

- **`Tag`** — A uint identifying the DICOM tag in error
- **`AttributeSet`** — the MCattributeSet derived class containing the attribute in error
- **`ValueNumber`** — specifies which of the attribute's values was in error
- **`ErrorDescription`** — a String describing the error
- **`ErrorNumber`** — a number identifying the error. Refer to the MCvalidationError class in the Assembly Windows Help File for the error numbers that may be returned.

The **ToString** method in the MCvalidationError class provides a convenient way to display a validation error. A sample string (that includes imbedded line feeds) follows:

Example Output

```
Attribute tag:      0x00100010 (Patient's Name)
Dataset:            Mergecom.MCdataSet
Value Number:       0
Description:        Invalid value for this tag's VR
Error Number:       28
```

It is on the initial call to the `validate` method that all the validation takes place and that the results of the validation for the entire message are logged to the message log file. Subsequent calls to the `getNextValidationError` method simply steps through the results of the validation, passing additional errors found back to the application. A sample log file report follows.

Example Log File

```
01-11 13:52:09.00 7919 MC3 T5: (0008,0005) VI: Unable to check condition
01-11 13:52:09.00 7919 MC3 T5: (0008,0023) VI: Unable to check condition
01-11 13:52:09.00 7919 MC3 T5: (0008,0033) VI: Unable to check condition
01-11 13:52:09.00 7919 MC3 T5: (0010,1010) VE: [41Y ] Invalid value for this tag's VR
01-11 13:52:09.00 7919 MC3 T5: (0018,0010) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0018,0015) VE: Required attribute has no value
01-11 13:52:09.00 7919 MC3 T5: (0018,0020) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0018,0021) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0018,0022) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0018,0023) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0018,0050) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0018,0080) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0018,0081) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0018,0082) VW: Invalid attribute for service
```

```
01-11 13:52:09.00 7919 MC3 T5: (0018,0084) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0018,0085) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0018,0091) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0018,1041) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0018,1060) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0018,1250) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0018,5101) VE: Required attribute has no value
01-11 13:52:09.00 7919 MC3 T5: (0020,0032) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0020,0037) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0020,0052) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0020,0060) VI: Unable to check condition
01-11 13:52:09.00 7919 MC3 T5: (0020,1040) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0020,1041) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0028,0006) VI: Unable to check condition
01-11 13:52:09.00 7919 MC3 T5: (0028,0030) VW: Invalid attribute for service
01-11 13:52:09.00 7919 MC3 T5: (0028,0034) VI: Unable to check condition
01-11 13:52:09.00 7919 MC3 T5: (0028,1101) VI: Unable to check condition
01-11 13:52:09.00 7919 MC3 T5: (0028,1102) VI: Unable to check condition
01-11 13:52:09.00 7919 MC3 T5: (0028,1103) VI: Unable to check condition
01-11 13:52:09.00 7919 MC3 T5: (0028,1201) VI: Unable to check condition
```

Notice in this log file that all warnings and informational messages are also logged. This is always the case, although the first violation returned to the application was an error because `MCvalidationLevel.Full` was specified. The message log agrees in that the first VE (Validation Error) logged is for the attribute Patient's Age (0010,1010). The log states that the message contains "`41Y `" as the value for this attribute. Part 6 of DICOM clearly states that this attribute has a value representation of AS (Age String) and part 5 states that for this VR the value should have a leading zero and be represented as "`041Y`". There is also one other error flagged in this message. The required attribute View Position (0018,5101) had no value.

The Overhead of Validation

Performance Tuning

DICOM attribute set validation does involve processing overhead. The most significant overhead is in the accessing of the message info files, and significantly less overhead is involved in actually validating the contents of the message structure. It is **important** to understand that depending on the way in which your MCdimseMessage, MCfile or MCdataSet object was created, this validation overhead can occur at different points in your application; see *Table 20*.

Table 20: Point of performance overhead associated with attribute set validation.

Message Object Construction Method	Point at which file access overhead for validation occurs
new MCdimseMessage if command/service supplied	new MCdimseMessage
new MCdimseMessage if no command/service supplied	MCdimseMessage.validate Note: You must use setServiceCommand method before validating and/or sending a message created in this manner.

Message Object Construction Method	Point at which file access overhead for validation occurs
new MCdimseMessage(MCdataSet) if command/service supplied when constructing MCdataSet	new MCdataSet()
new MCdimseMessage(MCdataSet) if no command/service supplied when constructing MCdataSet	MCdimseMessage.validate
MCdimseMessage = read::MCassociation	MCdimseMessage validate
new MCfile if command/service supplied	new MCfile
new MCfile if no command/service supplied	MCfile.validate Note: You must use setServiceCommand method before validating and/or sending a message created in this manner.
new MCfile (MCdimseMessage) if command/service supplied when constructing MCdimseMessage	new MCdimseMessage ()
new MCfile (MCdimseMessage) if no command/service supplied when constructing MCdataSet	MCfile.validate
new MCdataSet If command/service supplied	new MCdataSet
new MCdataSet If no command/service supplied	MCdataSet.validate

When the attribute set is constructed by providing the service and command to be used, there is an up-front performance cost but it provides additional validation as you set the value of attributes in the message object. When the service and command are not known at construction time, the cost occurs when the `validate` call is made.

Performance Tuning

Many times the `validate` method is selectively used in an application: as a runtime option or conditionally compiled into the source code. Validation might only be used during integration testing or in the field for diagnostic purposes. Reasons for this include performance since the overhead associated with message validation may be an issue, especially for larger messages having many attributes or on lower-end platforms. Also, validation can clutter the message log with warnings and errors that may not be desirable in a production environment. Performance issues related to message handling are discussed further under Message Exchange later in this document.

Validating a Single Attribute

validateAttribute

If you wish to validate only a single attribute, you may use the **`validateAttribute`** method of the MCdimseMessage, MCfile or MCdataSet class. The `validateAttribute` method works exactly as the validate method with the exception that you provide a tag parameter to identify the attribute use wish to have validated.

```
MCdimseMessage msg;  // non-null reference

bool validates;
validates = msg.validateAttribute(0x00100010,
       MCvalidateLevel.Errors_Only);
if (!validates)
{
    MCvalidationError err = msg.getNextValidationError();
    while (err != null)
    {
        System.Console.Out.WriteLine(err.ToString());
        err = msg.getNextValidationError();
    }
}
```

Streaming Attribute Sets

Message stream defined

When DICOM messages are exchanged over a network, they are in an encoded format specified by the DICOM standard and the negotiated transfer syntax. Merge DICOM Toolkit calls this encoded format a **message stream** and supplies powerful methods that allow your applications to work directly with message streams.

When your application builds or parses attribute sets as described earlier, it works with the MCattributeSet objects contained in MCdimseMessage or MCfile objects. These MCattributeSet objects abstract and encapsulate the DICOM message and hides its details from the developer. When you send a DICOM message over the network, Merge DICOM internally creates a DICOM message stream that is passed over the network. This message stream is an encoded stream of bytes that follows all the rules of DICOM.

Streaming methods

streamOut
streamIn
streamLength
MCstreamOffset

Merge DICOM Toolkit also supplies methods to generate and read DICOM message streams directly (see *Figure 13*). The methods are available in the MCdimseMessage class and the MCattributeSet class. The **`streamOut`** method creates a message stream from the contents of an MCdimseMessage object, while the **`streamIn`** method populates an MCdimseMessage object from a message stream. Also, **`streamLength`** method is supplied to calculate the length of the DICOM stream that would result from using the `streamOut` call.

(The `streamLength` method is also provided in the MCfile class to return the actual length of the streamed file object. And the `streamIn`, `streamOut` and `streamLength` methods are also provided by the MCattributeSet class, so any attribute set may be used to create a stream.)

The streamIn methods return the byte offset from the beginning of the stream to the next attribute after the stop tag parameter.

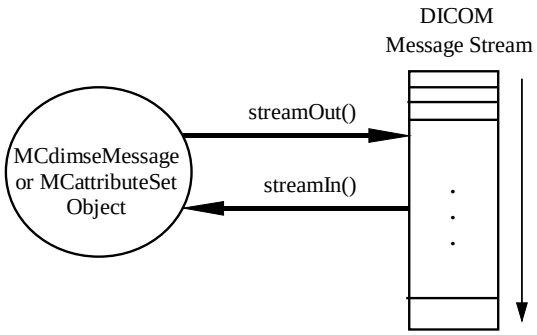

Figure 13: Relationship between an Attribute Set Object and a Message Stream

Creating a message stream

MCstreamOutCallback MCdata receiveData method

A `streamOut` call could look like the following:

```
class MyStreamHandler : MCdataSink {
    …
    public void receiveData(MCdata data, System.Object
        origin)
    {
        // Store the data described in data
        // as appropriate
        …
        if (errorOccured)
            throw new MCcallbackCannotComplyException();
    }
}

MCdimseMessage msg;   // non-null reference
MyStreamHandler streamHandler = new MyStreamHandler();
try {
    msg.streamOut(0x00080000, 0x7FDFFFFF,
        MCtransferSyntax.ExplicitLittleEndian,
        streamHandler);
}   catch (MCnoAttributesException e) {…}
    catch (MCillegalArgumentException e) {…}
    catch (MCcallbackCannotComplyException e) {…}
```

This call converts the attributes from (0008,0000) through (7FDF,FFFF) in the MCdimseMessage object identified by `msg` into a DICOM message stream using the explicit VR little endian transfer syntax. The message stream will be encoded using the explicit VR little endian transfer syntax.

`streamHandler` is an instance of MyStreamHandler, a class that implements the **MCdataSink** interface. The MCdataSink interface requires a **`receiveData`** method that receives and manages the stream data a block at a time. See the API description in the Assembly Windows Help File for further details.

Retrieving message data from a message stream

Once your application has done the above and stored the stream somewhere, you could later rebuild a message object containing only group 0008 using:

MCstreamInCallback MCdata provideData method

```
class MyStreamHandler : MCdataSource {
    …
    public MCdata provideData(bool isFirst, Object origin)
    {
        // Retrieve a portion (or all) of the stream
        // and return it as an MCdata object
        …
        if (errorOccured)
            throw new MCcallbackCannotComplyException();
    }
}

MCdimseMessage msg;   // non-null reference
MyStreamHandler streamHandler = new MyStreamHandler();

try {
    msg.streamIn(0x00080000, 0x0008FFFF,
        MCtransferSyntax.ExplicitLittleEndian,
        streamHandler);
} catch (MCexception e) {…}
```

`streamHandler` is an instance of MyStreamHandler, a class that implements the **MCdataSource** interface. The MCdataSource interface requires a **`provideData`** method that retrieves the stream data a block at a time and returns it to the Merge DICOM library. This call converts only the attributes in group 0008 of the stream supplied by your MyStreamHandler callback class and places them in the message identified by `msg`. It is important that the transfer syntax specified in this call is identical to that used to create the stream or the call will fail with an error.

Performance Tuning

The same kind of performance issues apply in the callback classes discussed when retrieving pixel data. Namely, your settings of LARGE_DATA_STORE and OBOW_BUFFER_SIZE should take into consideration the capabilities of your platform.

Why use message streams?

Message streams can be very valuable to your application for debugging and validation purposes. By writing DICOM message streams out to a binary file, you have a compact and reproducible representation of a message. You can directly examine the binary message stream to see how the data would be sent over the network. Also, you can read this binary file in again later to reconstruct the original message object. Once you have the message object you can use the usual toolkit methods to examine or alter its contents.

Message to XML conversion

Converting message data to an XML string

writeToXML method

The MCabstractMessage provides a **writeToXML** method which can be utilized to convert its derived classes (MCfile and MCdimseMessage) to an XML string.

You can convert a list of attributes of an MCfile or MCdimseMessage, along with their values into an XML string by using the **writeToXML** method of the MCabstractMessage. The writeToXML method creates an XML string describing the contents of the MCfile or MCdimseMessage. The XML buffer will be written to the stream identified by the stream object provided.

Note: If MCfile or MCdimseMessage objects contain an attribute with a Value Representation of SQ (sequence of items), each item in the sequence is converted into its XML representation.

The following example shows how the writeToXML method is utilized at a high level.

```
MCfile myFile; // a non-null file reference
MCxmlOptions xmlOptions = MCxmlOptions.XmlOptIncludeBulks |
      XmlOptExcludeSequences;
StreamWriter writer = new StreamWriter( "myFile" );
// convert DICOM file to an XML file
myFile.writeToXML( writer, xmlOptions );
writer.Close();
```

The following configuration flags are defined in the MCxmlOptions enumeration and are available for the MCabstractMessage to XML conversion:

```
// Use the default settings
XmlOptDefault                  = 0x0
// Store bulk attributes (VR is OB or OW) in the XML
XmlOptIncludeBulks             = 0x1
// Store Pixel Data buffer in the XML
XmlOptIncludePixelData         = 0x2
// Do not store Sequence attributes in the XML
XmlOptExcludeSequences         = 0x4
// Do not store Private attributes in the XML
XmlOptExcludePrivateAttributes = 0x8
// Use Base64 encoding for bulks and UN VR attributes
XmlOptBase64Binary             = 0x10
```

XML to Message conversion

Converting an XML string into a message

readFromXML method

The MCabstractMessage provides a **readFromXML** method that can be utilized to read attribute values from an XML string into MCabstractMessage's derived classes (MCfile and MCdimseMessage).

The content of the message is not cleared before processing XML attributes. The existing attributes in the message are overridden if they are present in the XML string.

The following example shows how the readFromXML method is utilized at a high level.

```
StreamReader reader = new StreamReader( "myXMLFile" );
MCfile file = new MCfile();
file.readFromXML( reader );
reader.Close();
```

Message Exchange (Network Only)

We have discussed how associations are managed as well as how messages objects are populated and parsed. Now we discuss how these DICOM messages are exchanged with other application entities over the network.

The exchange of DICOM messages between AE's only occurs over an open association. After the DICOM client (SCU) application opens an association with a DICOM server (SCP), the client sends request messages to the server application. For each request message, the client receives back a corresponding response from the server. The server waits for a request message, performs the desired service, and sends back some form of status to the client in a response message. This process, along with the corresponding Merge DICOM Toolkit method calls, are pictured in *Figure 14*.

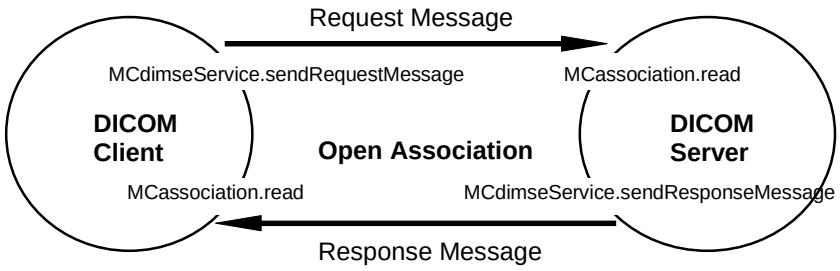

Figure 14: Message Exchange in Merge DICOM Toolkit Applications

Reading Network Messages

Reading messages

MCassociation.read

The **`read`** method of the MCassociation class is always used to retrieve the next message available on the network connection. It returns an MCdimseMessage object that encapsulates the DICOM message. Its only parameter is a timeout value:

```
public MCdimseMessage read(long timeout);
```

The *`timeout`* parameter specifies, in milliseconds, how long your process will wait for a message before the `read` call times out and returns control to your application code. The thread handling your association will be blocked during this waiting period and the system processor will be available for other threads. Setting *`timeout`* to 0 is equivalent to polling, since `read` returns immediately, whether a message has been received or not. A *timeout* of -1 indicates wait forever, or until a message arrives, before returning. An MCtimeoutException will be thrown if the time expires before a message arrives.

Using the MCdimseService

Sending messages

sendRequestMessage

sendResponseMessage

To send request messages you use the **sendRequestMessage** method of the MCdimseService class, and to send response messages you use the **sendResponseMessage** method. You should note, however, that you will probably be using a sub-class of the MCdimseService class and those derived classes usually provide other methods to send messages. This section describes the use of the MCdimseService class directly.

You must relate each instance of the MCdimseService class with a specific association by passing an MCassociation reference as a parameter to the class constructor:

```
MCassociation myAssoc;  // a non-null reference
MCdimseService myService = new MCdimseService(myAssoc);
```

Using the sendRequestMessage method

Requesting a DICOM service

There are four forms of the MCdimseService `sendRequestMessage` method:

```
public void sendRequestMessage(MCdimseMessage msg)
public void sendRequestMessage(MCdimseMessage msg, String
       affectedSopInstanceUID)
public void sendRequestMessage(String metaServiceName,
       MCdimseMessage msg)
public void sendRequestMessage(String metaServiceName,
       MCdimseMessage msg, String affectedSopInstanceUID)
```

At minimum, the `sendRequestMessage` call must provide a reference to an MCdimseMessage object that encapsulates the message to be sent (the *msg* parameter).

Some DICOM SOP Classes require that you assign an Affected SOP Instance UID to the composite message object being sent; in those cases, you must use the *affectedSopInstanceUID* parameter, providing the UID.

The *metaServiceName* parameter is also optional. If it is null or it if is an empty string, it will be ignored. Some DICOM services (e.g., the Basic Print Service) allow you to support multiple meta services. Each meta service consists of a set of basiC/C++ DICOM services. In some cases a DICOM application may support multiple meta services over the same association. When two of the meta services include the same basic service, this *metaServiceName* parameter is used to tell Merge DICOM which meta service to use when sending the message.

Using the sendResponseMessage method

Responding to a DICOM service request

There are two forms of the MCdimseService `sendResponseMessage` method:

```
public void sendResponseMessage(MCdimseMessage requestMsg,
       short statusCode)
public void sendResponseMessage(MCdimseMessage requestMsg,
       MCdimseMessage responseMsg, short statusCode)
```

This `sendResponseMessage` method allows you to respond to a message received from the remote application. It is called after a successful MCassociation.read call. The *requestMsg* parameter identifies which DICOM message is being responded to.

The *statusCode* parameter provides the status of the requested operation and must be a valid response code for the service involved. Response codes are defined in MCdimseSerivce. Many DICOM services do not require a response of more than just a status. Others, however, (e.g., C_FIND_RSP) require the setting of several message attributes.

The *responseMsg* parameter, if provided, must have been constructed for the same service as the *requestMsg*, and for an appropriate command. For example, if responding to a C-STORE request message, the *responseMsg* would be constructed as follows:

```
responseMsg = new MCdimseMessage
      (MCdimseService.C_STORE_RSP,
      requestMsg.ServiceName );
```

Response codes for specifiC/C++ DICOM commands are described in the Assembly Windows Help File and in Part 4 of the DICOM Standard. Constants for the various request and response codes are defined in the MCdimseService class.

If *responseMsg* is not provided, Merge DICOM will create one automatically.

Merge DICOM Toolkit handles the command set for you

Some DICOM services require that values for certain command set attributes (i.e. group 0 attributes) be set. Merge DICOM automatically adds command set attributes when an MCdimseMessage is constructed. With the exceptions listed below, you must set any command set attribute values before sending the message. However, this method will set the following attribute values for you – **if you have not set them**:

- The group length attribute (0000,0000) value is always set by Merge DICOM Toolkit.
- If the message command requires it, the Affected SOP Class UID attribute (0000,0002) value is set to the service's abstract syntax UID.
- The command attribute (0000,0100) value is always set by Merge DICOM Toolkit.
- The Response Message ID attribute (0000,0120) value is set to the message ID of the last received Request message for this association.
- The Data Set Type attribute (0000,0800) value is always set by Merge DICOM Toolkit.

Using Attribute Containers

Attribute values can be stored in your own container class

MCattributeContainer

The **MCattributeContainer** is the interface for classes that will provide methods for the library to get and set a given attribute's value. A class which implements this interface is registered with the library using the **`registerAttributeContainer`** method of the MCapplication class. The library considers this interface a "container" for a specific attribute's value and calls methods of the class to get or set the attribute's value. Such container classes are registered only for attribute's that have values of great length, such as pixel data.

The methods of the MCattributeContainer class exhibit one significant difference from the methods used in the `MCdataSink` and `MCdataSource` interfaces described earlier. MCattributeContainer classes 'throttle' the data flow as the message object is communicated over the network. Rather than storing attributes with large OB/OW/OF values within the message object itself, your application is responsible for maintaining the value of these attributes.

We will also see that MCattributeContainer objects affect accessing media files. See the discussion of this in the DICOM Files section later in this manual.

Using an Attribute Container in a Server Application

Server callbacks

A server (SCP) application can register a MCattributeContainer object that will be called repetitively as the attribute's value arrives on an association during a MCassociation.read call. By the time the `read` method returns to the application, the attribute value will already have been handled by your MCattributeContainer class. The MCattributeContainer class could be used by the server to treat this large block of OB/OW/OF data (usually pixel data) specially (e.g., store in a frame buffer, filter through decompression hardware, write to disk...) without any overhead introduced by the MCdimseMessage object.

Using an Attribute Container in a Client Application

Client callbacks

A client (SCU) application can register a MCattributeContainer object that will be called repetitively as the attribute's value is transmitted over an association during an MCdimseService class `sendRequestMessage` or `sendResponseMessage` call. During either of these calls, the attribute value will be handled by your registered MCattributeContainer object before these calls can return to your application. The MCattributeContainer class can also be used by the client to specially manage OB/OW/OF data (e.g., read from a frame buffer, filter through compression hardware or software, read from disk...) without any overhead introduced by the MCdimseMessage object.

Declaring an MCattributeContainer Class

How to declare an MCattributeContainer class

The MCattributeContainer interface requires that your container class provide five methods that will be called by the Merge DICOM library at different times. A sample class declaration follows:

```
public class MyContainer : MCattributeContainer{
    …
    public uint provideDataLength(MCattributeSet attribSet,
        MCtag tag) {
    }

    public MCdata provideData(MCattributeSet attribSet,
        MCtag tag, bool isFirst) {
    }

    public void receiveDataLength(MCattributeSet attribSet,
        MCtag tag, uint dataLength) {
    }

    public void receiveData(MCattributeSet attribSet, MCtag
        tag, MCdata data, bool isFirst) {
    }

    public void receiveMediaDataLength( MCattributeSet
        attribSet, MCtag tag, uint dataLength, uint
        dataOffset) {
    }
}
```

Writing the provideDataLength method

Merge DICOM Toolkit calls for the attribute's data length

The provideDataLength method is called by the library to request the data length of the attribute identified by the *tag* parameter. The attribute set containing the attribute is identified by the *attribSet* parameter. The library will call this method before it begins calling the provideData method.

This method is required to return the length of the attribute's value. The returned data length must be an even number. The hex value 0xffffffff (which means undefined length in DICOM)may be returned if the attribute contains encapsulated data.

If the method cannot comply with the request, it must throw an **MCcallbackCannotComplyException**. If the exception is thrown, Merge DICOM will make no further calls for this instance of the attribute.

```
public uint provideDataLength(MCattributeSet attribSet,
        MCtag tag)
{
    uint length;
    // If unable to get the attribute value's length
        throw new MCcallbackCannotComplyException();
    // Set length to the number of bytes contained
    // in the attribute's value.
    return length;
}
```

Writing the provideData method

Merge DICOM Toolkit calls for the attribute's data value

The provideData method is called by the library to request a portion of the attribute's value. The attribute is identified by the *attribSet* and *tag* parameters.

The value is returned in an MCdata object that contains the data buffer (managed or unmanaged), the *Length* property giving the amount of data in the buffer, and a bool indicator (*IsLast*). The buffer must contain an even number of bytes and may be empty. The *IsLast* property of the returned MCdata object must be set to `true` if this is the last portion of the value that will be provided. Merge DICOM will no longer call `provideData` for this instance of the attribute after *IsLast* is returned `true`. The *Length* property must be set to the number of significant bytes in the data buffer.

isFirst is set by the library to `true` if this is the first request for the attribute's value.

If the method cannot comply with the request, it must throw an **MCcallbackCannotComplyException**. If the exception is thrown, Merge DICOM will make no further calls for this instance of the attribute.

```
public MCdata provideData(MCattributeSet attribSet,  MCtag
      tag, bool isFirst)
{
   byte[] array = new byte[4096]; bool isLast = false; uint
      size = 0;

   if (unableToProvideData)
      throw new MCcallbackCannotComplyException();

   if (isFirst) {
      // Perhaps open up the data source
   }

   // Read the next portion of the value into array
   // put length read into size
   if (thereIsNoMoreData)
      isLast = true;
   MCdata data = new MCdata(array,size);
   data.IsLast = isLast;
   return data;
}
```

Writing the receiveDataLength method

Merge DICOM Toolkit calls to provide the attribute's data length

The `receiveDataLength` method is called by the library to provide the callback class with the data length (*`dataLength`* parameter) of the attribute identified by the *`attribSet`* and *`tag`* parameters. The library calls this method before calling the receiveData method.

If the method cannot comply with the request, it must throw an **`MCcallbackCannotComplyException`**. If the exception is thrown, Merge DICOM will make no further calls for this instance of the attribute.

```
public void receiveDataLength(MCattributeSet attribSet,
       MCtag tag, uint dataLength)
{
    if (dataLengthUnacceptable)
    throw new MCcallbackCannotComplyException();

    mYlocalLength = dataLength;
}
```

Writing the receiveData method

Merge DICOM Toolkit calls to provide the attribute's data length

The `receiveData` method is called by the library to provide the callback with some or all of the attribute's value. The library has set the *`data`* field of the MCdata object to a byte array containing all or a portion of the attribute's value.

`isFirst` will be `true` if the library is presenting the first portion of the attribute's value. If this is the last portion of the value that the library will present, the *`IsLast`* property of the MCdata object will be `true`.

The data buffer in the MCdata object may have a length of zero if this is the last portion of the data (i.e. `MCdata.IsLast = true`).

Note that this method is not called when processing the MCmediaStorageService `readFileBypassLargeData` call because the OB/OW/OF data is left on the media. Instead, the library calls the `receiveMediaDataLength` method (see below).

If the method cannot comply with the request, it must throw an **`MCcallbackCannotComplyException`**. If the exception is thrown, Merge DICOM will make no further calls for this instance of the attribute.

```
public void receiveData(MCattributeSet attribSet, MCtag
       tag, MCdata data, bool isFirst)
{
    if (anyProblemOccurs)
       throw new MCcallbackCannotComplyException();
    if (isFirst) {
    // Perhaps open an output data sink
    }

    // Save the data

    if (data.IsLast) {
    // perhaps close the data sink
    }
}
```

Merge DICOM Toolkit calls to provide the attribute's data length and data offset

Writing the receiveMediaDataLength method

The `receiveMediaDataLength` method is called by the library when it is reading a file from media while it is processing an MCmediaStorageService `readFileBypassLargeData` call. This method provides the total size of the attribute's value and the byte offset of the attribute's value from the beginning of the media file.

Note: The library calls this method instead of calling the `receiveData` method when it is processing the MCmediaStorageService `readFileBypassLargeData` call.

If the method cannot comply with the request, it must throw an **`MCcallbackCannotComplyException`**. If the exception is thrown, Merge DICOM will make no further calls for this instance of the attribute.

```
public void receiveMediaDataLength(MCattributeSet
      attribSet, MCtag tag, uint dataLength,  uint
      dataOffset)
{

   if (anyProblemOccurs)
      throw new MCcallbackCannotComplyException();
   // Perhaps save the dataOffset and dataLength
   // so they can be used later to access the data
}
```

Registering Your MCattributeContainer

How to register an MCattributeContainer object

registerAttributeContainer

Each Application Entity registers its own MCattributeContainer objects. The `MCapplication` class **`registerAttributeContainer`** method is used to register an MCattributeContainer object to be used with the Application Entity:

```
MCapplication myApp;  // a non-null reference
MyContainer myContainer = new MyContainer();
MCattributeContainer oldContainer;
oldContainer = myApp.registerAttributeContainer(new
      MCtag(MCdicom.PIXEL_DATA), myContainer);
```

This call registers `myContainer`, an instance of the MyContainer class (which must implement the MCattributeContainer interface). `myContainer` will handle the pixel data (7FE0,0010) attribute for `myApp`. A single MCattributeContainer object can be multiply registered to handle many tags. Also, a single MCattributeContainer object will handle both transmittal and reception of the data associated with the tag(s). If the return value is `null` you know that there was not a previously-registered attribute container for the attribute.

Releasing Your MCattributeContainer

How to release an MCattributeContainer object

releaseAttributeContainer

To "de-register" your attribute container class, use the **`releaseAttributeContainer`** method of the MCapplication class. The method releases the callback object that was registered for the attribute identified by the MCtag parameter. The callback's methods will no longer be called when the attribute's value is being received or when the attribute's value is required.

The callback's methods will still be called, however, for **MCdimseMessage** objects or **MCfile** objects that were created before this method call was made.

Sequences of Items

Values for SQ attributes

MCitem

The DICOM Value Representation SQ is used to indicate a DICOM attribute that contains a value that is a sequence of items. Each item in the sequence is an attribute set (**MCitem** class). Each of the attribute sets can also contain attributes that have a VR of SQ. This powerful capability allows the nesting of attribute sets, or the definition of 'container' objects (such as folders, film boxes, directories, etc.).

Items in a message stream

Figure 15 shows a DICOM message containing a sequence of items running two levels deep. Note that these nested sequences are contained **within** the same Message Stream. Sequences of items can also be contained in a DICOM file, and we will see that they are contained in DICOMDIR's. An attribute whose value is a sequence of items is simply an attribute that has a potentially large and complex value. Fortunately, Merge DICOM Toolkit allows your application to deal with sequences of items an item at a time and hierarchically, as pictured in *Figure 15*, and takes care of the encoding of the sequence within the DICOM message stream.

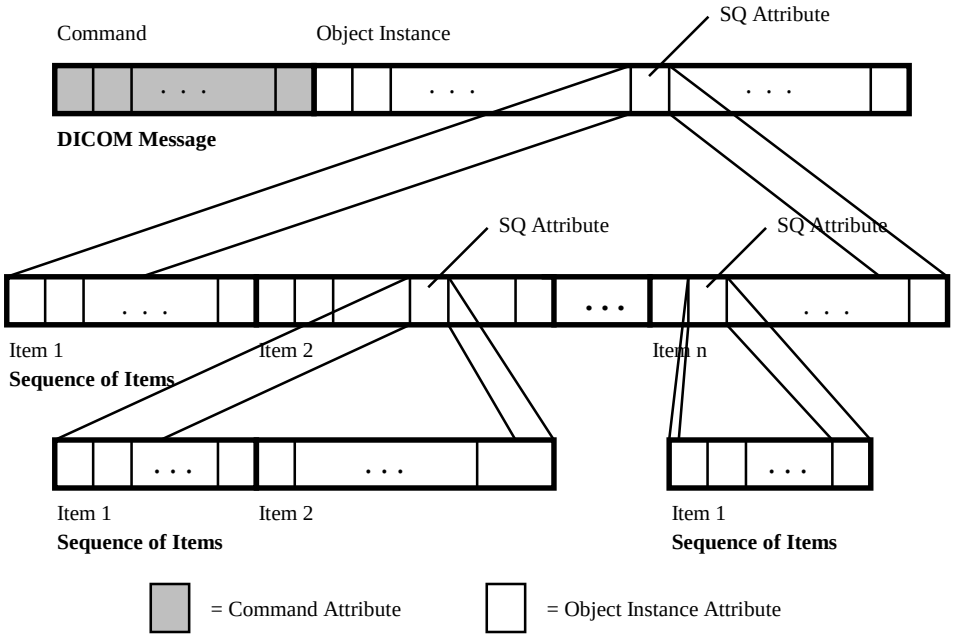

Figure 15: A DICOM Message containing doubly nested sequences of items.

Encoding and decoding attributes in an item

Each item in a sequence is handled as a special derived sub-class of the MCattributeSet class, called **MCitem**. All the MCattributeSet methods are inherited by the MCitem class.

Similar to the constructors for MCdataSet objects, there are two variations of the MCitem constructor. You can create an "empty" item attribute set by providing no parameter to the constructor. If you wish to have the item populated by Merge DICOM with specific attributes, you supply an `itemName` parameter used to identify the item. If the `itemName` is unknown to Merge DICOM a warning message is logged and an empty MCitem object is constructed.

```
MCitem emptyItem = new MCitem();
MCitem myItem = new MCitem("REF_FILM_BOX");
```

Available item names are listed in the `message.txt` file for attributes in messages having a VR of SQ. The contents of each item are also listed in the `message.txt` file. Below are two excerpts of `message.txt`, one showing a reference to the Referenced Film Box Item, and the other the contents of that item.

```
BASIC_FILM_SESSION - N_SET_RQ

0008,0005    Specific Character set                    CS    3
Defined Terms:  ISO-IR 100, ISO-IR 101, ISO-IR 109, ISO-IR 110,
ISO-IR144, ISO-IR 127, ISO-IR 126, ISO-IR 138, ISO-IR 148
0008,0012    Instance Creation Date                    DA    3
0008,0013    Instance Creation Time                    TM    3
0008,0014    Instance Creator UID                      UI    3
0008,0016    SOP Class UID                             UI    3
0008,0018    SOP Instance UID                          UI    3
2000,0010    Number of Copies                          IS    3
2000,0020    Print Priority                            CS    3
Enumerated Values: HIGH, MED, LOW
2000,0030    Medium Type                               CS    3
Defined Terms:     PAPER, CLEAR FILM, BLUE FILM
2000,0040    Film Destination                          CS    3
Enumerated Values: MAGAZINE, PROCESSOR
2000,0050    Film Session Label                        LO    3
2000,0060    Memory Allocation                         IS    3
2000,0500    Referenced Film Box Sequence              SQ    3
Item Name(s): REF_FILM_BOX

                                   .
                                   .
                                   .
```

```
Item Name: REF_FILM_BOX

0008,1150    Referenced SOP Class UID                    UI    1
0008,1155    Referenced SOP Instance UID                 UI    1
```

Encoding items in a sequence

To encode an item into an attribute of Value Representation SQ, treat the attribute as a multi-valued attribute, where each value is an MCitem object. This means using an MCitem reference with the MCattributeSet addValue, setValue or indexer. Similarly, the MCattributeSet indexer methods return MCitem objects when you request the value of a sequence attribute.

The following sample code fragment gives an example of encoding a Pre-formatted Grayscale Image Item into a sequence:

```
MCitem myItem = new MCitem("PREFORMATTED_GRAYSCALE_IMAGE");

MCdataSet ds;  // non-null reference to the
               // data set we are building

myItem[MCdicom.PIXEL_ASPECT_RATIO, 0] = "1";
myItem[MCdicom.PIXEL_ASPECT_RATIO, 1] = "1";

/* encode other item attributes here */

             .
             .
             .
/* now add the item to the sequence */
ds[MCdicom.PREFORMATTED_GRAYSCALE_IMAGE_SEQUENCE, 0] =
      myItem;
             .
             .
             .
```

DICOM Files

DICOM message objects

MCfile
MCdir
MCfileMetaInfo

Maintaining a DICOM file set is a matter of maintaining various DICOM files and a single DICOM directory file (DICOMDIR).

DICOM media files are encapsulated in the **MCfile** class. A sub-class of the MCfile class, the **MCdir** class, encapsulates a special DICOM directory file, called the **DICOMDIR**. Just as DICOM network messages (the MCdimseMessage class) contain a command set (MCcommandSet) and a data set (MCdataSet), so the MCfile class contains a special file meta information attribute set (the **MCfileMetaInfo** class) and a data set (MCdataSet).

Figure 16 demonstrates the attribute sets contained in DICOM network messages and DICOM file objects.

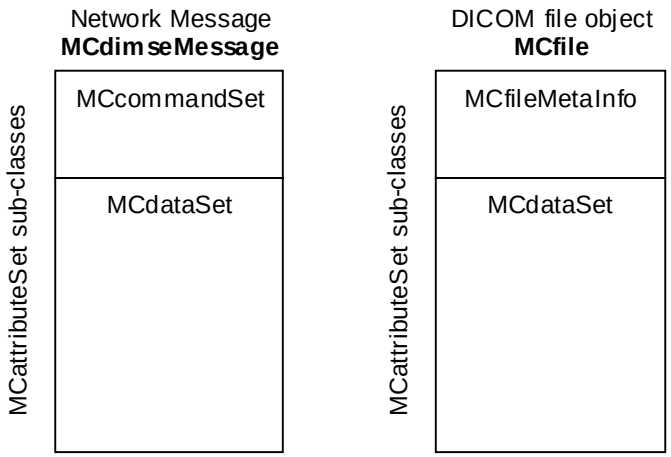

Figure 16: Attribute Sets Contained in DICOM objects

Constructing a new MCfile Instance

Several ways to construct file objects

The Media Storage Service manipulates MCfile objects. There are several options available to construct new instances of the MCfile class. As mentioned before, each MCfile instance contains an MCdataSet object and an MCfileMetaInfo object. The MCfile may be constructed with a pre-populated data set or with an empty data set.

It is **important** to realize that constructing an MCfile object does not create the physical DICOM file out on the media; the write method of the MCmediaStorageService class described later does that.

Construct an MCfile object with a pre-populated data set

Populating the file with a service's attributes

Two forms of the constructor create an MCfile object that contains all of the attributes of a DICOM file that will be used for the given *service* and *command*. The attributes are maintained in a MCfileMetaInfo object and an MCdataSet object. Normally you will only deal with the data set and the file meta information attributes will be set automatically by Merge DICOM Toolkit.

```
MCfile myFile = new MCfile(command, serviceName,
       "MyFileName");
or-
MCfile myFile = new MCfile(command, serviceName);
```

serviceName and *command* are used to access configuration information that describes the attributes of the message. If such configuration information is not available, an empty file object is created, and a warning message is logged. The filename parameter, if used, provides the name of the operating system file to be associated with this MCfile object

Construct an MCfile object with an empty data set

Constructing an 'empty' file object

Two forms of the constructor are used if the service and command are not yet known, or if there is no need to validate that values will be set only for attributes assigned to a given service/command pair. It creates an empty MCdataSet object. The resulting MCfile object is not associated with any particular DICOM service or command. If the **`validate`** method is to be called, the **`setServiceCommand`** method must be called first to associate this file object with a given DICOM service and command.

```
MCfile myFile = MCfile("MyFileName");
or-
MCfile myFile = new MCfile();
```

The filename parameter, if used, provides the name of the operating system file to be associated with this MCfile object.

Performance Tuning

Just as when you construct an empty MCdimseMessage object for networking, when you construct an empty MCfile object, the message info and data dictionary files are not accessed. This object contains no pre-allocated attributes in the contained MCfileMetaInfo and MCdataSet objects, and the **`setServiceCommand`** method must be called to set the service and command for this file before it can be written to the file set. As in the case of networking, this approach is more efficient but penalizes you in the area of run-time error checking.

Convert an MCdimseMessage object to an MCfile object

Preparing to save a network message to file

Another form of the constructor converts a network message object into a file object associated with a specified file system file.

```
MCdimseMessage message;  // a non-null reference
MCfile myFile = MCfile(message, "MyFileName");
```

The data set contained in *message* will be used in this object.

Note: The original MCdimseMessage and the new MCfile objects will be sharing the same MCdataSet object. The filename parameter provides the name of the operating system file to be associated with the MCfile object.

Accessing the service and command properties

Service and Command Properties

setServiceCommand
ServiceName
Command

It the service and command for the MCfile object were not specified when the object was constructed, they can be provided later, using the **`setServiceCommand`** method. The service and command must be set if you wish to use the validate method. The **`Command`** and **`ServiceName`** properties can be used to retrieve these properties.

```
myFile.setServiceCommand("STANDARD_CT",
      MCdimseService.C_STORE_RQ);
String serviceName = myFile.ServiceName;
ushort command = myFile.Command;
```

Working with the contained file meta information

File Meta Information Attribute Set

MCfileMetaInfo
MetaInfo

The File Meta Information (MCfileMetaInfo) encapsulated in the MCfile class contains identifying information about the data set also encapsulated in a DICOM file. The meta information consists of a fixed-length 128-byte file Preamble, a DICOM Prefix ("DICM"), followed by several DICOM attributes providing the properties of the encapsulated data set. (Refer to Part 10 of the DICOM standard for more details.) The contents of this object are maintained automatically by Merge DICOM Toolkit, although the **`MetaInfo`** property of the MCfile class returns a reference to its contained **MCfileMetaInfo** object. Using that reference, you can call the methods it inherits from MCattributeSet.

```
// Get the file meta info attribute set
MCfileMetaInfo metaInfo = myFile.MetaInfo;

// Retrieve the attributes of the file meta info
foreach (MCattribute attr in metaInfo.Attributes) {
      // Use MCtag toString method to list each tag
   System.Console.Out.WriteLine(attr.Tag.ToString());
}
```

File Meta Information Attribute Set

MCfileMetaInfo
MetaInfo
Preamble property

Accessing the File Preamble

The **`Preamble`** property is provided to access the preamble portion of the file meta info. The property can be used to get or set the preamble. This property is available in both the MCfile container class and the MCfileMetaInfo class.

```
byte[] preamble = myFile.MetaInfo.Preamble;

// The same could be accomplished as follows:
byte[] preamble = myFile.Preamble;
```

The Data Set

MCdataSet
DataSet
removeFileValues

Working with the contained data set

The data set (MCdataSet) encapsulated in the MCfile class contains the DICOM information object associated with the file. You can retrieve a reference to the contained MCdataSet object with the **`DataSet`** property. Using that reference, you can call the methods it inherits from MCattributeSet.

```
// Get the data set attribute set
MCdataSet ds = myFile.DataSet;

// Retrieve the attributes of the data set
foreach (MCattribute attr in ds.Attributes) {
    // Use MCtag toString method to list each tag
    System.Console.Out.WriteLine(attr.Tag.ToString());
}
```

Starting Over

removeFileValues

Resetting the MCfile object

You can use the **`removeFileValues`** method to remove the values of each attribute in the file's data set and meta info set. This is equivalent to calling the MCattributeSet `removeValues` method for each attribute in the file's contained MCdataSet and MCfileMetaInfo objects. This method is useful for applications that reuse an MCfile object and want to insure there are no attribute values remaining from the last use of the object. Attributes with no values are skipped when streaming the collection for network transfer or for writing a DICOM file.

```
myFile.removeFileValues();
```

File validation

Data Set Validation

validate
validateAttribute

You can validate that the data set meets the requirements of the service/command pair associated with the file by using the **`validate`** method of the MCfile object. (You could also use the validate method of the contained MCdataSet object). You can also validate an individual attribute within the data set using the **`validateAttribute`** method.

```
MCfile myFile;  // non-null reference

bool validates;
validates = myFile.validateAttribute(MCdicom.PATIENTS_NAME,
      MCvalidateLevel.Errors_Only);
if (!validates)
{
   MCvalidationError err = myFile.g,etNextValidationError();
   while (err != null)
   {
      System.Console.Out.WriteLine(err.ToString());
      err = msg.getNextValidationError();
   }
}
```

To understand the overhead involved in file validation, please refer to The Overhead of Validation on page 109.

The MCfile stream

How big is the file stream?

streamLength

Merge DICOM concatenates the contents of the MCfileMetaInfo object and the MCdataSet object when streaming the MCfile object. We will discuss later how to use the MCmediaStorageService to read and write the DICOM file streams. You may need to know ahead of time how large the streamed file object will be before writing the object to media. The stream's size can be obtained using the **`streamLength`** method.

```
uint length = myFile.streamLength();
```

Setting the file transfer syntax UID

The transfer syntax UID

TransferSyntax property

You can use the **`TransferSyntax`** property to set the value of the DICOM "Transfer Syntax UID" associated with this file. It is sets attribute (0002,0010) in the file's file meta information to the UID string you provide.

```
myFile.TransferSyntax =
      MCtransferSyntax.ExplicitLittleEndian;
```

Setting the file system file associated with the MCfile object

The file's name

FileName property

Whether or not you specify a file system file name when constructing the MCfile object, you can set or get the name at any time using the **`FileName`** property:

```
myFile.FileName = "FileName";
String file = myFile.FileName;
```

Listing the file's attributes

Listing the file

list

The **`list`** method produces a report describing the contents of the File Meta Info and Dataset contained in this MCfile. The report will be written to the stream identified by the stream object provided. If no stream object is specified the report will be written to the system's standard output (stdout).

If the object contains an attribute with a Value Representation of SQ (sequence of items), each item in the sequence will be listed. Each sequence of items is indented in the listing four spaces to the right of its owning message or items.

```
System.IO.StreamWriter writer = new StreamWriter("myFile");
myFile.list(writer);  // list to myFile

myFile.list();   // list to stdout
```

Using the MCmediaStorageService Class

DICOM
File Service Classes

MCfileService
MCmediaStorageService

Analogous to the MCdimseService class that handles DICOM network message exchange, the **MCmediaStorageService** class handles services dealing with DICOM media files. The MCmediaStorageService provides methods to read and write DICOM files.

All the media interchange functionality of the DICOM Toolkit relies on methods that you supply to interface with the particular physical medium and file system format on your target device. This approach was chosen because of the wide variety of media and file system configurations allowed by the DICOM Standard and the potentially unlimited combination of media devices, device drivers, and file system combinations for which DICOM media interchange applications may be developed.

You must interface with the selected media device

Similar to working with streams, if your application will read DICOM files, you must use a class that implements the MCdataSource interface. If your application will write DICOM files, you must use a class that implements the MCdataSink interface. These interfaces have been discussed in detail in previous sections.

You will find that the DICOM Toolkit provides powerful DICOM media functionality by supplying your application with:

- a greatly simplified way to deal with the complex encoding and decoding required within a DICOM file.
- an API that is very consistent with that used for the maintenance of DICOM messages used in network functionality; many of the encoding and decoding methods already described apply equally to DICOM file attribute sets.

To perform all this functionality on your medium of choice, you need only supply the two file system interface implementation, just discussed, and use the methods of the MCmediaStorageService class.

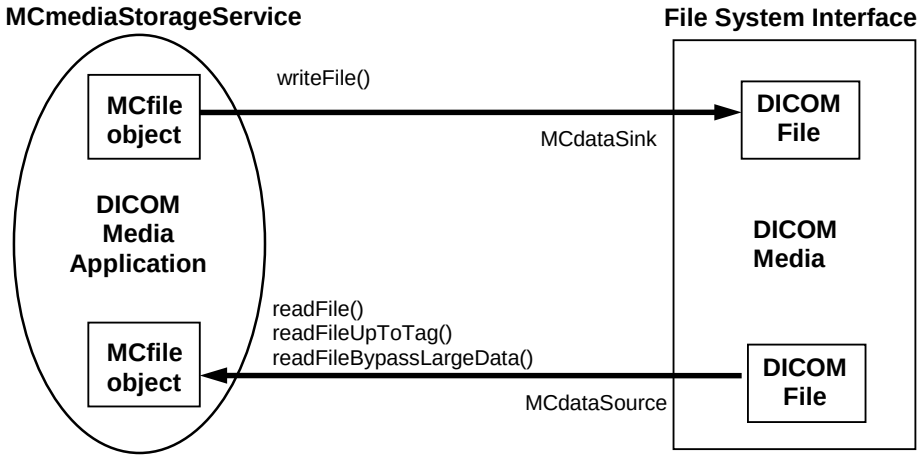

Figure 17: Classes and Methods Used for Handling DICOM Media Files

Constructing an MCmediaStorageService object

A Media Storage Service is associated with your application

Each MCmediaStorageService instance is associated with an application entity. It is necessary to provide a reference to your AE's MCapplication object when constructing an instance of the MCmediaStorageService.

```
MCapplication myApp;  // non-null reference
MCmediaStorageService myMediaService = new
       MCmediaStorageService(myApp);
```

Reading Files

Three ways To read a media file

To read in the contents of a DICOM file for analysis or parsing you use one or three methods available in the MCmediaStorageService class for reading DICOM files. Each of the methods passes a reference to a class that implements the MCdataSource interface. The callback class will actually read the file stream from media.

The three methods are:

Pass OB/OW/OF data to registered callbacks (if any)

readFile

- **readFile** — This method calls your MCdataSource to retrieve the DICOM file stream. It decodes the stream and populates the MCfileMetaInfo and MCdataSet attribute sets that are contained in your MCfile object. If the stream contains an attribute for which an MCattributeContainer instance has been registered, the attribute's value is passed on to the MCattributeContainer, rather than having Merge DICOM store the value.

```
MCapplication myApp;  // non-null reference

MCfile fileObj = new MCfile("FileName");
MCdataSource source = new
       MCfileDataSource(fileObj.FileName);

MCmediaStorageService service =  new
       MCmediaStorageService(myApp);
try {
   service.readFile(fileObj, source);
} catch (Exception e) {…}
```

Pass OB/OW/OL data offset and length only to registered callbacks (if any)

readFileBypassLargeDat

- **readFileBypassLargeData** — This method reads the file just as the readFile method does, with one exception. If the stream contains an attribute for which an MCattributeContainer instance has been registered, the attribute's value is NOT passed on to the MCattributeContainer, but instead the values length and offset into the file is passed to the receiveMediaDataLength method of the MCattributeContainer. This provides the opportunity for substantial performance improvement. Note that if no MCattributeContainer is registered for the OB/OW/OF attribute, the attribute's value will be stored by Merge DICOM Toolkit, as usual.

```
MCapplication myApp;  // non-null reference

MCfile fileObj = new MCfile("FileName");
MCdataSource source = new
       MCfileDataSource(fileObj.FileName);

MCmediaStorageService service = new
       MCmediaStorageService(myApp);
try {
   service.readFileBypassLargeData(fileObj, source);
} catch (Exception e) {…}
```

Read only up to the Pixel Data

readFileUpToTag

- **readFileUpToTag** — This method retrieves the values of the file stream just as the readFile method does, but it will stop requesting data when it has processed the last attribute whose tag is ≤ the "last tag" parameter. It will return the file offset to the first byte of the first attribute whose tag is > the "last tag" parameter.

  This method can be used to increase performance for handling attributes of Value Representations OB, OW, or OF. It is most useful when a file contains pixel data (7FE0, 0010) as its last attribute and this pixel data is very large. In these instances you may wish to ignore the pixel data, read it in later (using the returned file offset), or process it directly from the file using your own special filters or hardware. This can be done by specifying MCdicom.PIXEL_DATA for "last tag" parameter.

```
MCapplication myApp;  // non-null reference

MCfile fileObj = new MCfile("FileName");
MCdataSource source = new
       MCfileDataSource(fileObj.FileName);

MCmediaStorageService service = new
       MCmediaStorageService(myApp);
try {
   service.readFileUpToTag (fileObj, MCdicom.PIXEL_DATA,
       source);
} catch (Exception e) {…}
```

See the Assembly Windows Help File for a detailed description of the use of these read methods.

Creating and Writing Files

writeFile

When you have a populated MCfile object you can create a DICOM file stream by using the **`writeFile`** method of the MCmediaStorageService class. This method utilizes the MCdataSink class to present the file stream for writing to media.

If your application has one or more MCattributeContainer objects registered for OB/OW/OF attributes, the `writeFile` method retrieve an attribute's value from a callback if an MCattributeContainer callback is registered for it.

DICOM allows you to pad the file

The second parameter of `writeFile` specifies a byte padding number. The attribute (FFFC, FFFC) will be added to the MCfile object and given a length such that the total length of the streamed file is a multiple of the byte padding number. If 0 is specified, there will be no padding of the file stream. The parameter must be an even number.

Merge DICOM Toolkit assures accurate group lengths

If the file contains "group length" attributes (i.e. attributes with tags of the form gggg0000: any group, element zero), this method will automatically calculate the group length value when supply it to the *callback*.

You control the endian used in the file stream

The byte stream will be formatted in the transfer syntax specified by the attribute transfer syntax UID (specified by the MCdicom.TRANSFER_SYNTAX_UID (0002,0010) attribute in the file's Meta Information set). (You can set this value using the `TransferSyntax` property of the MCfile class.)

Merge DICOM Toolkit automatically sets fields in the File Meta Information

Two group 2 attributes within the file meta information will be automatically filled in **if you have not set them yourself**:

- The Implementation Class UID (0002,0012) will be filled in with the value set for the IMPLEMENTATION_CLASS_UID configuration value in the mergecom.pro file.
- The Implementation Version Name (0002,0013) will be filled in with the value set for the IMPLEMENTATION_VERSION configuration value in the mergecom.pro file.

Sample media write snippet

The following example is taken from the StorageSCP sample application.

```
MCapplication myApp;  // non-null reference
MCdimseMessage msg;  // a network message just received
MCfile fileObj = new MCfile(msg, "MyFileName")
MCdataSink destination = new
       MCfileDataSink(fileObj.FileName);
MCmediaStorageService mediaService = new
       MCmediaStorageService(myApp);
try {
   mediaService.writeFile(fileObj, 0, destination);
} catch (Exception e) {
   System.Console.Out.WriteLine("writeFile failed", e);
}
```

The DICOMDIR file

Review: what are directory entities? directory records?

As discussed earlier, in each DICOM File Set (containing many DICOM files) their must exist a single DICOM File with the reserved File ID "DICOMDIR". This file contains identifying information for the file set that most often includes a directory of the file sets contents. A media interchange application would make use of and maintain the DICOMDIR to locate a particular file within the file set for processing.

Structure

A information object portion of a DICOMDIR file has a special structure that is described in Part 3 (PS 3.3) of the DICOM Standard. We described the this structure earlier in this document (see *Figure 8* on page 29) as a hierarchy of **directory records**, where each directory record may contain a set of related directory records. These directory records can have a one-to-one relationship to a DICOM file within the file set described by the DICOMDIR. Directory records do not have to reference a DICOM file, they can be used solely to contain information that helps an application navigate down the directory hierarchy to locate the desired DICOM file.

As an example, the Root directory record might contain two Patient directory records and a Topic directory record. One of the Patient directory records references multiple Series records and a Film Session record for that Patient. Each of these Series records reference Image records for that patient. It is these Image records that reference the DICOM file containing the image objects acquired for the Patient whose directory hierarchy we have traversed. (See above on page 29 for a description of the allowed entity hierarchies).

This directory record hierarchy is encoded within the DICOMDIR as a single, potentially very complex, sequence of items where each item is a directory record. Byte offset attributes within the directory records are used to point to other directory records at the same level in the hierarchy, as well as lower-level directory records. DICOM File ID's are encoded in the directory record if the record references a particular DICOM file in the file set.

DICOMDIR's are ugly!

The key observation here is that rather than using nested Sequences of Items to encode the DICOMDIR hierarchy, the standard chose to use a single, potentially very large, sequence of items and byte offsets. The standard defines these byte offsets as being measured "from the first byte of the file meta-information". As you might well imagine, the complexity of maintaining these byte offsets accurately, as directory records are added to or removed from directory entities within the DICOMDIR file, is very great and can be very cumbersome.

But we pretty them up

Fortunately, the Merge DICOM Toolkit supplies methods that make DICOMDIR maintenance much simpler for your application. These methods are now described.

Constructing a new MCdir Instance

MCdir extends the MCfile class

MCdir

To create a special type of DICOM file that contains a DICOMDIR directory, construct a new MCdir object. The MCdir class is a subclass of the MCfile class since it is simply a form of a DICOM file. The file name provided MUST refer to a file named "DICOMDIR".

```
MCdir myDICOMDIR = MCdir ("HERE/DICOMDIR");
```

When you construct an MCdir object, Merge DICOM creates an instance of the MCfile class that has a service name of "DICOMDIR" and uses C_STORE_RQ for the command (see *Table 5*).

The MCdirRecord class

Directory record properties

MCdirRecord

The MCdirRecord class represents a DICOMDIR record and is simply a container for a number of public properties and methods:

- **`Parent`** — A property that returns the MCdirRecord instance for the parent record of this directory record.
- **`RecordName`** — This property gets a String name assigned to the requested DICOMDIR record
- **`Directory`** — This property returns the MCdir that this directory record is contained within.
- **`IsLast`** — a Boolean that is true if the record is the last child record of the parent record. This property is not updated when new child records are added or removed to/from the parent record.
- **`RecordItem`** — This property is the MCitem instance for the directory record.
- **`getChildCount()`**— This method returns a count of the number of child records.
- **`getFirstChild()`**— This method returns the MCdirRecord for the first child of this directory record.
- **`getNextChild()`**— This method can be called repeatedly to get subsequent child directory records.
- **`getReferencedRecordCount()`** — This method traverses all child directory records and returns a count of these records.
- **`addChildRecord(String newItemName)`** — This method adds a child record.
- **`delete()`** — This method deletes the current directory record within the DICOMDIR.
- **`deleteChildren()`** — This method deletes all the child directory records below this record.

The following sections describe these methods and properties in further detail.

Navigating the DICOMDIR

navigating through a DICOMDIR

MCdirRecord
MCdir.Root

The MCdirRecord class provides routines for traversing the DICOMDIR. The MCdir class has a property, **`Root`**, which returns an MCdirRecord instance which is a placeholder for the parent of the root directory records within the DICOMDIR. This directory record in turn can be used to navigate through the root records of the DICOMDIR and the remainder of the DICOMDIR.

The first step in navigating a DICOMDIR usually involves getting the **`Root`** property root of the DICOMDIR. From there the **`getFirstChild`** and **`getNextChild`** are used to traverse lower level records referenced by a particular record. When working with a specific directory record, the **`RecordItem`** property can be used to get the MCitem associated with a specific directory record. This MCitem instance can be used to access the attributes within the directory record. The following code sample shows the use of these routines.

```
MCdir dir;  // Non-null reference
MCdirRecord rootRec = dir.Root;
MCdirRecord curRec;
curRec = rootRec.getFirstChild();
while (curRec != null)
{
    if (curRec.RecordName.Equals("DIR_REC_PATIENT"))
    {
        MCitem item = curRec.RecordItem;
        // Access patient directory record tags here
    }
    curRec = rootRec.getNextChild();
}
```

In the above example, the directory records below the root record could be traversed by calling the getFirstChild and getNextChild routines.

Please refer to the MCdirRecord and MCdir classes in the Assembly Windows Help File for further details on traversing a DICOMDIR.

Adding and Deleting DICOMDIR Records

Adding and deleting directory records

dirAddRecord
dirDeleteRecord

The addition and deletion of directory records are handled using the **addChildRecord**, **deleteChild**, and **delete** methods. These calls are prototyped as follows:

```
public MCdirRecord addChildRecord( String newItemName);
public void deleteChildren();
public void delete();
```

When adding a directory record, you must supply a string identifying the directory record item type (*`newItemName`*). For example, if you wished to add a Study record, *`newItemName`* would be a string containing "`DIR_STUDY`". The method returns an MCdirRecord object encapsulating the newly created directory record.

Automatic deletion of referenced items

When deleting records using **deleteChildren** or **delete** no parameters are required. When a directory record is deleted, all lower level directory entities (and the directory records contained within them) are also freed. The deleteChildren deletes all children for the current directory record. The delete method deletes the current record and all of the children of the directory record.

Make sure you are committed!!

The Merge DICOM Toolkit updates and maintains all the byte offsets that are part of the DICOMDIR structure automatically. But, one important note: All the changes to a DICOMDIR are made in memory and are not committed to media until a `writeFile` call is made.

Memory Management

Performance Tuning

Really! It's not a memory leak

The Merge DICOM **C library** contains it's own memory management routines that are optimized for how it uses memory. They have been adapted to manage specific data structures that are frequently allocated by the Merge DICOM C toolkit. These include but are not limited to data structures for associations, messages, and tags. The memory management routines have the characteristic that they do not actually "free" the memory that has been acquired. Instead, they mark the data as being free and place the memory in a list for reuse later. These routines have been optimized to quickly acquire and free memory being used by Merge DICOM Toolkit. They also allow Merge DICOM to not depend on the memory management of a particular operating system.

These memory routines have also been extended for use with variable sized memory buffers. Merge DICOM uses these routines to allocate buffers in sizes between 4 bytes and 28K. When an allocation is requested, Merge DICOM will take the smallest buffer that will fit the bytes requested. These buffers will be kept in Merge DICOM Toolkit's internal memory pool and never freed. For allocations larger than 28K, Merge DICOM will simply use the 'C' functions `malloc()` and `free()`.

The end result of these routines is that applications using Merge DICOM expand to the maximum amount of memory used at one time. The total memory allocation will not shrink from this point. In applications that repeatedly perform a consistent operation, the memory being used by Merge DICOM should stabilize and not increase in size. As a result of these routines, the first time an application performs a DICOM operation is typically slower than subsequent operations.

When developing a DICOM application with Merge DICOM Toolkit, the most memory intensive operation is dealing with image data. The following sections discuss various Merge DICOM methods. A description is given of how these methods manage memory in conjunction with various Merge DICOM configuration settings.

Assigning Pixel Data

`The MCattribute.setValue method` is used in conjunction with the `MCdataSource` interface to assign OB, OW, or OF data to a DICOM attribute. These value representations are used to store image data or other large data elements. `setValue` is described in further detail elsewhere in this manual.

The `MCdataSource` implementation can pass data to `the library` in a single call, or in several smaller chunks. When passed data, `the library` will

Performance Tuning

setValueFromMethod

allocate a buffer the size of the chunk passed to it and copy the data into this buffer for storage.

The size of data returned by provideData will dictate how the image data is stored. If the data is passed in chunks smaller than 28K, Merge DICOM Toolkit's internal memory management code will be used. If the chunks are larger than 28K, `malloc`() will be used to allocate storage for the buffers. If large images are being dealt with, it may be desirable to pass this data in chunks larger than 28K, so the memory is freed after processing has been completed for the image. This will keep the nominal memory usage of Merge DICOM lower. When passing data in chunks less than 28K, it is recommended that sizes of 16K, 20K, 24K, or 28K be used. Using these size chunks will reduce the overhead in storing the data.

`The library` can also be directed to store data in temporary files. The `LARGE_DATA_STORE` and `LARGE_DATA_SIZE` configuration options in the mergecom.pro file dictate when data is stored in temporary files. When the `LARGE_DATA_STORE` option is set to `FILE`, data elements that are larger than configured by the `LARGE_DATA_SIZE` option are stored in temporary files. The size of buffer returned by provideData does not have an effect on memory usage.

Using Attribute Containers

Merge DICOM also supplies a method to allow the user to manage image data through the use of registered callback methods. The MCapplication class `registerAttributeContainer` method associates a callback class with a DICOM attribute such as pixel data. These callbacks are limited to attributes with the value representations of OB, OW, or OF. When encountered, the attribute's data is passed to a method of the registered callback class instead of being stored within Merge DICOM Toolkit. The callback is also used to supply the attribute's data. The size of data elements to use in callbacks can also be specified. The `CALLBACK_MIN_DATA_SIZE` configuration option can specify the minimum size or length required for the use of a registered callback class.

There are three models in which `registerAttributeContainer` can be used. First, it can be used to seamlessly replace Merge DICOM Toolkit's memory management functions. Use of this method can for the most part be hidden from the application. Secondly, the method can be used as an interface to receive or supply data only when it is needed. When writing a network application, the image data can be supplied to the user directly as it is read off the network. The data can also be supplied when it is about to be written to the network. This functionality can also be used when creating and reading DICOM files. Finally, `registerAttributeContainer` can be used to save an image to disk as it is received over the network.

This can be tricky

Replacing Merge DICOM Toolkit's Memory Management Functions

When using `registerAttributeContainer` to replace Merge DICOM Toolkit's memory management functions, the user would still use the MCattribute `readBulkData` and `setValue` methods to access the image. When requested, Merge DICOM will receive or supply the attribute's value to the attribute container.

Performance Tuning

getValueToMethod
setValueFromMethod

Accessing Data When Needed

When dealing with large multi-frame images, it is sometimes impractical to load the entire image into memory at once. `registerAttributeContainer` can be used to access image data only when needed. The memory requirements of an application can be greatly reduced by using this functionality.

Performance Tuning

read
sendRequestMessage

When reading messages from the network, the MCassociation `read` method supplies the user's registered callback class with the image data. If the data does not need to be byte swapped into the system's native endian, the amount of data supplied with each call is dictated by the PDU size of the data received. When the data is byte swapped, the length of data is specified by the `WORK_BUFFER_SIZE` configuration value. As the data is received, it would typically be written to disk in this scenario. When the `read` method returns, the user is given the message read from the network encapsulated in an MCdimseMessage object. The message object still contains a link to the registered callback class. This link can be removed by calling the MCattributeSet `removeValues` method for the registered attribute. The header data can then be examined and later written to disk.

When sending data over the network, the MCdimseService class `sendRequestMessage` method (or the equivalent method from one of the MCdimseService subclasses) will call the user's registered callback class for the image data. The data can be supplied to Merge DICOM in any length as required by the user's application. The data is typically read from disk at this point and directly passed to Merge DICOM Toolkit. After `sendRequestMessage` receives the data, it byte swaps the data if needed, and then writes it to the network.

This functionality is conducive to storing a message's header data separately from its image data. Depending on system requirements, this may be an aid in quickly loading image data while bypassing Merge DICOM Toolkit. The complete image file can be reassembled later using Merge DICOM Toolkit.

Performance Tuning

Read streamOut

Saving Received Images Directly to Disk

In conjunction with the registered callback class, data can also be stored directly to disk when it is being read. The image's header data can be written to disk from within the registered callback. The user must write the attribute tag, value representation if needed, and the length of the image data attribute to the file. The image data is written to the file in subsequent calls to the user's registered callback method.

When `read` is parsing a message being received, it will notify the user's registered callback class when it has parsed the header information and determines the image data's length. The registered callback's `receiveDataLength` method will be called, providing the length of the registered attribute's value, as well as a reference to the MCattributeSet being populated. At this point, the user can stream the header file to disk using the MCattributeSet `streamOut` method. As the image data is received, it can be added to the end of this file.

Data can also be stored as DICOM files with this method. The message cannot be converted into a file object at this point using the special form of the MCfile constructor as would normally done. So, a separate MCfile object must be constructed to add the DICOM Part 10 Meta Header information. This header can be written out from within the callback by using the `streamOut` method on the contained MCfileMetaInfo object. After the end of the meta header, the message can be streamed to disk with a call to `streamOut` in the transfer syntax specified in the Meta Header. As subsequent image data is passed to the user's callback class, the data can be written to file. Because the endian of the transfer syntax being written may be different than the endian of the system being used, there may be a need for byte swapping of the pixel data in this implementation.

There is a potential risk with this implementation. Although the data elements after the pixel data in the current definition of the DICOM image types are not widely used, future versions may add data elements that will get wider acceptance among implementors.

DICOM Structured Reporting

The Merge DICOM Toolkit provides high-level functionality to handle DICOM Structured Report (SR) Documents. This functionality provides a simple way for encoding and decoding SR Document content by manipulating content items and their attributes instead of tags and values.

Structured Report Structure and Modules

The DICOM standard Part 3 defines the following generic types of SR Information Object Definitions (IODs):

- ***Basic Text SR Information Object Definition*** — The Basic Text Structured Report (SR) IOD is intended for the representation of reports with minimal usage of coded entries (typically used in Document Title and headings) and a hierarchical tree of headings under which may appear text and subheadings. Reference to SOP Instances (e.g., images or waveforms or other SR Documents) is restricted to appear at the level of the leaves of this primarily textual tree. This structure simplifies the encoding of conventional textual reports as SR Documents, as well as their rendering.

- ***Enhanced SR Information Object Definition*** — The Enhanced Structured Report (SR) IOD is a superset of the Basic Text SR IOD. It is also intended for the representation of reports with minimal usage of coded entries (typically Document Title and headings) and a hierarchical tree of headings under which may appear text and subheadings. In addition, it supports the use of numeric measurements with coded measurement names and units. Reference to SOP Instances (e.g., images or waveforms or SR Documents) are restricted to display at the leaf level of this primarily textual tree. The Enhanced Structured Report (SR) IOD enhances references to SOP Instances with spatial regions of interest (points, lines, circle, ellipse, etc.) and temporal regions of interest.

- ***Comprehensive SR Information Object Definition*** — The Comprehensive SR IOD is a superset of the Basic Text SR IOD and the Enhanced SR IOD which specifies a class of documents (the content of which may include textual and a variety of coded information, numeric measurement values, references to the SOP Instances and spatial or temporal regions of interest within such SOP Instances). Relationships by-reference are enabled between Content Items.

There are more specific SR IODs defined in the DICOM, like ***Key Object Selection Document*** and ***Mammography CAD SR***. These IODs use the same method to encode data but differ in constrains on the Content Item Types and their relationships. *Figure 18* illustrates the typical SR Document structure. The top level header is similar to the DICOM image IODs and consists of the same Patient, Study and Series modules. The main difference from other IODs is the ***SR Document Content Module***. The attributes in this Module convey the content of an SR Document.

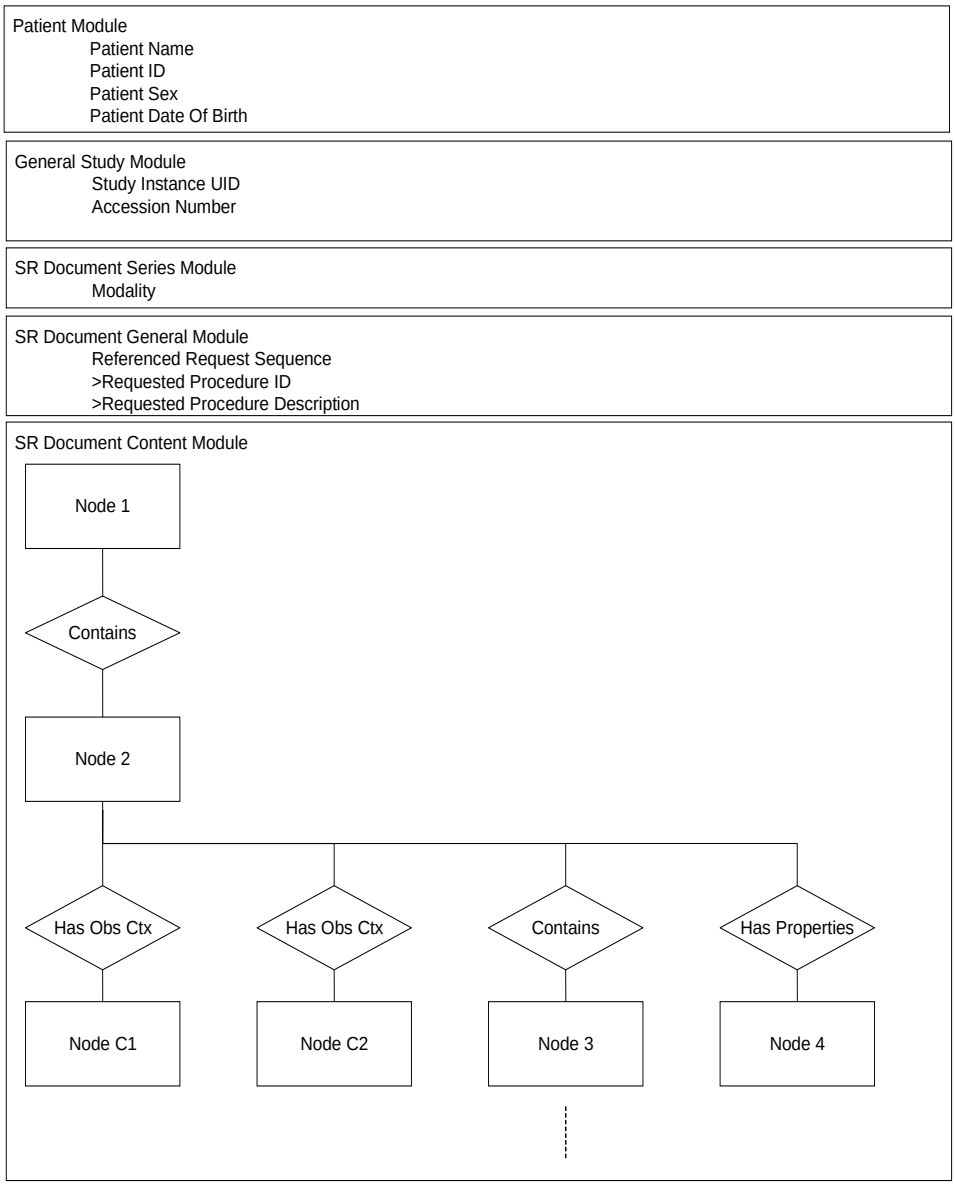

Figure 18: SR Document Structure example

The SR Document Hierarchy

The Document Content Module has a tree structure and consists of a single root Content Item (Node 1) that is the root of the SR Document tree. The root Content Item conveys either directly or indirectly, all of the other nested Content Items in the document. The hierarchical structuring of the Content Tree is provided by recursively nesting Content Items. A parent (or source) Content Item has an explicit relationship to each child (or target) Content Item, and is conveyed by the Relationship Type. *Figure 19* illustrates the relationship of SR Documents to Content Items and the relationships of Content Items to other Content Items, as well as to the Observation Context.

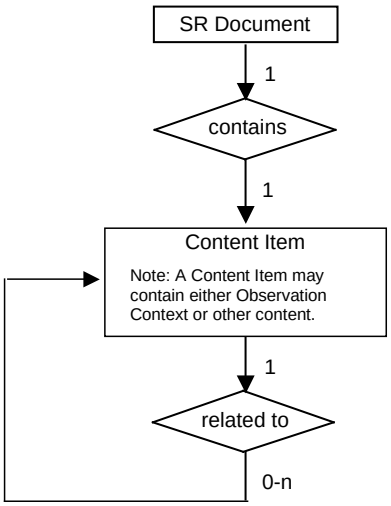

Figure 19: SR Information Model

Each Content Item contains the following.

- A name/value pair, consisting of:
  - a single Concept Name Code that is the name of a name/value pair or a heading; and
  - a value (text, numeric, code, etc.);
- References to images, waveforms or other composite objects, with or without coordinates; and
- Relationships to other Items, either by-value through nested Content Sequences, or by-reference.

Note: Some Content Item Types can have multiple values.

Content Item Types

Table 21 defines all possible Content Item Types that can be used in the SR Document Content Module. The choice of which may be constrained by the IOD in which this Module is contained. The Merge DICOM Toolkit Class column specifies the enumerated value used in the Toolkit to identify the Content Item Type.

Table 21: SR Content Item Types

Item Type	Merge DICOM Toolkit Class	Concept Name	Description
TEXT	MCtextItem	Type of text, e.g., "Findings", or name of identifier, e.g., "Lesion ID"	Free text, narrative description of unlimited length. May also be used to provide a label or identifier value.
NUM	MCnumItem	Type of numeric value or measurement, e.g., "BPD"	Numeric value fully qualified by coded representation of the measurement name and unit of measurement.
CODE	MCcodeItem	Type of code, e.g., "Findings"	Categorical coded value. Representation of nominal or non-numeric ordinal values.
DATETIME	MCdateTimeItem	Type of DateTime, e.g., "Date/Time of onset"	Date and time of occurrence of the type of event denoted by the Concept Name.
DATE	MCdateItem	Type of Date, e.g., "Birth Date"	Date of occurrence of the type of event denoted by the Concept Name.
TIME	MCtimeItem	Type of Time, e.g., "Start Time"	Time of occurrence of the type of event denoted by the Concept Name.
UIDREF	MCuidReferenceItem	Type of UID, e.g., "Study Instance UID"	Unique Identifier (UID) of the entity identified by the Concept Name.
PNAME	MCpersonNameItem	Role of person, e.g., "Recording Observer"	Person name of the person whose role is described by the Concept Name.
COMPOSITE	MCcompositeItem	Purpose of Reference	A reference to one Composite SOP Instance which is not an Image or Waveform.
IMAGE	MCimageItem	Purpose of Reference	A reference to one Image. IMAGE Content Item may convey a reference to a Softcopy Presentation State associated with the Image.

Item Type	Merge DICOM Toolkit Class	Concept Name	Description
WAVEFORM	MCwaveformItem	Purpose of Reference	A reference to one Waveform.
SCOORD	MCspatialCoordinatesItem	Purpose of Reference	Spatial coordinates of a geometric region of interest in the DICOM image coordinate system. The IMAGE Content Item from which spatial coordinates are selected is denoted by a SELECTED FROM relationship.
TCOORD	MCtemporalCoordDateTimeItem MCtemporalCoordTimeOffsetsItem MCtemporalCoordPositionsItem	Purpose of Reference	Temporal Coordinates (i.e. time or event based coordinates) of a region of interest in the DICOM waveform coordinate system. The WAVEFORM or IMAGE or SCOORD Content Item from which Temporal Coordinates are selected is denoted by a SELECTED FROM relationship.
CONTAINER	MCcontainerItem	Document Title or document section heading. Concept Name conveys the Document Title (if the ONTAINER is the Document Root Content Item) or the category of observation.	CONTAINER groups Content Items and defines the heading or category of observation that applies to that content. The heading describes the content of the CONTAINER Content Item and may map to a document section heading in a printed or displayed document.

Relationship Types between Content Items

Table 22 describes the Relationship Types between Source Content Items and the Target Content Items. The choice of which may be constrained by the IOD in which this Module is contained. The Merge DICOM Toolkit Definition column specifies the enumerated value used in the Toolkit to identify the Content Item Relationship.

Table 22: SR Relationship Types

Relationship Type	Merge DICOM Toolkit Definition	Description
CONTAINS	CONTAINS	Source Item contains Target Content Item, e.g., CONTAINER "History" {CONTAINS: TEXT: "mother had breast cancer"; CONTAINS IMAGE 36}
HAS OBS CONTEXT	HAS_OBS_CONTEXT	Has Observation Context. Target Content Items shall convey any specialization of Observation Context needed for unambiguous documentation of the Source Content Item. e.g., CONTAINER: "Report" {HAS OBS CONTEXT: PNAME: "Recording Observer" = "Smith^John^^Dr^"}
HAS CONCEPT MOD	HAS_CONCEPT_MOD	Has Concept Modifier. Used to qualify or describe the Concept Name of the Source Content item, such as to create a post-coordinated description of a concept, or to further describe a concept. e.g., CODE "Chest X-Ray" {HAS CONCEPT MOD: CODE "View = PA and Lateral"} e.g., CODE "Breast" {HAS CONCEPT MOD: TEXT "French Translation" = "Sein"} e.g., CODE "2VCXRPALAT" {HAS CONCEPT MOD: TEXT "Further Explanation" = "Chest X-Ray, Two Views, Posteroanterior and Lateral"}
HAS PROPERTIES	HAS_PROPERTIES	Description of properties of the Source Content Item. e.g., CODE "Mass" {HAS PROPERTIES: CODE "anatomic location", HAS PROPERTIES: CODE "diameter", HAS PROPERTIES: CODE "margin", ...}.

Relationship Type	Merge DICOM Toolkit Definition	Description
HAS ACQ CONTEXT	HAS_ACQ_CONTEXT	Has Acquisition Context. The Target Content Item describes the conditions present during data acquisition of the Source Content Item. e.g., IMAGE 36 {HAS ACQ CONTEXT: CODE "contrast agent", HAS ACQ CONTEXT: CODE "position of imaging subject", ...}.
INFERRED FROM	INFERRED_FROM	Source Content Item conveys a measurement or other inference made from the Target Content Items. Denotes the supporting evidence for a measurement or judgment. e.g., CODE "Malignancy" {INFERRED FROM: CODE "Mass", INFERRED FROM: CODE "Lymphadenopathy",...}. e.g., NUM: "BPD = 5mm" {INFERRED FROM: SCOORD}.
SELECTED FROM	SELECTED_FROM	Source Content Item conveys spatial or temporal coordinates selected from the Target Content Item(s). e.g., SCOORD: "CLOSED 1,1 5,10" {SELECTED FROM: IMAGE 36}. e.g., TCOORD: "SEGMENT 60-200mS" {SELECTED FROM: WAVEFORM}.

Content Item Identifier

Content Items are identified by their position in the Content Item tree. They have an implicit order as defined by the order of the Sequence Items. When a Content Item is the target of a by reference relationship, its position is specified as the Referenced Content Item Identifier in the source Content Item. *Figure 20* illustrates an SR content tree and identifiers associated with each Content Item. The ***MCitemIdentifer*** class encapsulates the Content Item Identifer.

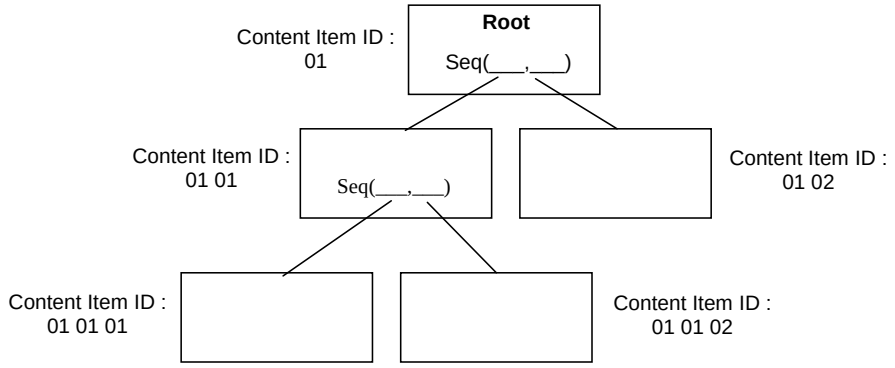

Figure 20: SR Item Identifier

Observation Context

Observation Context describes who or what is performing the interpretation, whether the examination of evidence is direct or quoted, what procedure generated the evidence that is being interpreted, and who or what is the subject of the evidence that is being interpreted.

Initial Observation Context is defined outside the SR Document Content tree by other modules in the SR IOD (i.e., Patient Module, Specimen Identification, General Study, Patient Study, SR Document Series, Frame of Reference, Synchronization, General Equipment and SR Document General modules). Observation Context defined by attributes in these modules applies to all Content Items in the SR Document Content tree and need not be explicitly coded in the tree. The initial Observation Context from outside the tree can be explicitly replaced.

If a Content Item in the SR Document Content tree has Observation Context different from the context already encoded elsewhere in the IOD, the context information of that Content Item shall be encoded as child nodes of the Content Item in the tree using the HAS OBS CONTEXT relationship, i.e., Observation Context is a property of its parent Content Item.

The context information specified in the Observation Context child nodes (i.e. target of the HAS OBS CONTEXT relationship) adds to the Observation Context of their parent node Content item, and shall apply to all by-value descendant nodes of that parent node regardless of the relationship type between the parent and the descendant nodes. Observation Context is encoded in the same manner as any other Content Item. Observation Context shall not be inherited across by-reference relationships.

Observation DateTime is not included as part of the HAS OBS CONTEXT relationship, and therefore is not inherited along with other Observation Context. The Observation DateTime Attribute is included in each Content Item which allows different observation dates and times to be attached to different Content Items.

The IOD may specify restrictions on Content Items and Relationship Types that also constrain the flexibility with which Observation Context may be described.

The IOD may specify Templates that offer or restrict patterns and content in Observation Context.

Structured Reporting Templates

Templates are patterns that specify the Concept Names, Requirements, Conditions, Value Types, Value Multiplicity, Value Set restrictions, Relationship Types and other attributes of Content Items for a particular application. SR Document templates are defined in the Part 16 of the DICOM Standard. Part 17 of the DICOM also has some explanatory information on encoding SR Documents. The Merge DICOM Toolkit SR Functions follow DICOM Templates structures and allow straightforward encoding based on template tables.

SR Templates are described using tables of the form shown on *Table 23*.

Table 23: SR Template Definition

	NL	Rel with Parent	VT	Concept Name	VM	Req Type	Condition	Value Set Constraint
1								
2								
3								

Row Number

Each row of a Template Table is denoted by a row number. The first row is numbered 1 and subsequent rows are numbered in ascending order with increments of 1. This number denotes a row for convenient description as well as reference in conditions. The Row Number of a Content Item in a Template may or may not be the same as the ordinal position of the corresponding node in the encoded document. The Merge DICOM Toolkit does not use this number in any way.

Nesting Level (NL)

The nesting level of Content Items is denoted by ">" symbols, one per level of nesting below the initial Source Content Item (of the Template) in a manner similar to the depiction of nested Sequences of Items in Modules Tables in Part 3 of the DICOM. When it is necessary to specify the Target Content Item(s) of a relationship, they are specified in the row(s) immediately following the corresponding Source Content Item. The Merge DICOM Toolkit provides functions to add nested (child) Content Items to the parent Content Item node. The following function shall be used to add a child node with relationship.

```
public void AddChild(MCcontentItem childItem,
      MCrelationshipType relationshipType)
```

Relationship with Source Content Item (Parent)

Relationship Type and Mode are specified for each row that specifies a target content item. The Relationship Types are enumerated in the *Table 22*.

Relationship Type and Mode may also be specified when another Template is included, either "top-down" or "bottom-up" or both (i.e., in the "INCLUDE Template" row of the calling Template, or in all rows of the included Template, or

in both places). There shall be no conflict between the Relationship Type and Mode of a row that includes another Template and the Relationship Type and Mode of the rows of the included Template.

When the relationship is defined in a form as R-RTYPE, it means that Relationship Mode is "By-reference "and Relationship Type is "RTYPE". For example, "R-INFERRED FROM". Merge DICOM Toolkit provides following functions to encode/decode references:

```
public void AddReference(MCitemRelationship reference)
public void RemoveReference(MCcontentItem targetItem)
public void RemoveReference(MCitemRelationship reference)
public ReadOnlyCollection<MCitemRelationship> References
```

Value Type (VT)

The Value Type field specifies the SR Value Type of the Content Item or conveys the word "INCLUDE" to indicate that another Template is to be included (substituted for the row). The Merge DICOM Toolkit provides specific classes for each Content Item Type as it described above.

Concept Name

Any constraints on Concept Name are specified in this field as defined or enumerated coded entries, or as baseline or defined context groups. Alternatively, when the VT field is "INCLUDE", the Concept Name field specifies the template to be included. The Merge DICOM Toolkit uses the ***MCbasicCodedEntry*** class to specify the Concept Name.

You will find that some of the Content Item types require Concep Name in the constructor and some are not, because it is optional for those Content Item Types. In that case, the Concept Name can be set by using the public property.

Templates define References to coded concepts take the following form:

```
EV or DT (ConceptNameValue, ConceptNameSheme,
      "ConceptNameMeaning")
```

For example, EV (T-04000, SNM3, "Breast") would mean that hardcoded values shall be used for that Concept Name. Some template items don't have DT or EV abbreviation and just specify the hardcoded values.

Abbreviations used in templates

The following abbreviations are used in template definitions.

- **EV** Enumerated Value —values for are provided in the brackets.
- **DT** Defined Term —values are provided in the brackets.
- **BCID** Baseline Context Group ID — identifier that specifies the suggested Context Group. The suggested values can be found in the DICOM Part 16 and identified by a Context ID provided in the brackets.
- **DCID** Defined Context Group ID — identifier that specifies the Context Group for a Coded Value that shall be used. The values can be found in the DICOM Part 16 and identified by a Context ID provided in the brackets.

- **BTID** Baseline Template ID — identifier that specifies a template suggested to be used in the creation of a set of Content Items. The referenced template can be found in the DICOM Part 16 and identified by a Template ID provided in the brackets.

- **DTID** Defined Template ID — identifier that specifies a template that shall be used in the creation of a set of Content Items. The referenced template can be found in the DICOM Part 16 and identified by a Template ID provided in the brackets.

Value Multiplicity (VM)

The VM field indicates the number of times that a Content Item of the specified pattern, or an included Template may appear in this position. *Table 24* specifies the values that are permitted in this field.

Table 24: Permitted Values for VM

Expression	Definition
i (where 'i' represents an integer)	Exactly i occurrences, where i>=1. e.g., when i=1 there shall be one occurrence of the Content Item in this position.
i-j	From i to j occurrences, where i and j are >=1 and j>i.
1-n	One or more occurrences

Requirement Type

The Requirement Type field specifies the requirements on the presence or absence of the Content Item or included Template. The following symbols are used.

- **M** — Mandatory. Shall be present.

- **MC** — Mandatory Conditional. Shall be present if the specified condition is satisfied.

- **U** — User Option. May or may not be present.

- **UC** — User Option Conditional. May not be present. May be present according to the specified condition.

Condition

The Condition field specifies any conditions upon which presence or absence of the Content Item or its values depends. This field specifies any Concept Name(s) or Values upon which there are dependencies.

References may also be made to row numbers (e.g., to specify exclusive OR conditions that span multiple rows of a Template table).

The following abbreviations are used.

- **XOR** — Exclusive OR One and only one row shall be selected from mutually-exclusive options.

Note: For example, if one of rows 1, 2, 3 or 4 may be included, then for row 2, the abbreviation "XOR rows 1,3,4" is specified for the condition.

- **IF** — Shall be present if the condition is TRUE; may be present otherwise.
- **IFF** — If and only if. Shall be present if the condition is TRUE; shall not be present otherwise.
- **CV** — Code Value
- **CSD** — Coding Scheme Designator
- **CM** — Code Meaning
- **CSV** — Coding Scheme Version

Value Set Constraint

Value Set Constraints, if any, are specified in this field as defined or enumerated coded entries, or as baseline or defined context groups.

The Value Set Constraint column may specify a default value for the Content Item if the Content Item is not present, either as a fixed value, or by reference to another Content Item, or by reference to an Attribute from the dataset other than within the Content Sequence (0040,A730).

Inclusion of Templates

A Template may include another Template by specifying "INCLUDE" in the Value Type field and the identifier of the included Template in the Concept Name field. All of the rows of the specified Template are in included in the invoking Template, effectively substituting the specified template for the row where the inclusion is invoked. Whether or not the inclusion is user optional, mandatory or conditional is specified in the Requirement and Condition fields. The number of times the included Template may be repeated is specified in the VM field.

We recommend that you implement templates as a subroutine or function call. In that case, the inclusion of the template will be implemented as a call to that template with passing parameters. Some of the templates defined in DICOM Part 16 already have predefined parameters and they are indicated by a name beginning with the character "$".

Overview of the Merge DICOM Toolkit SR Classes

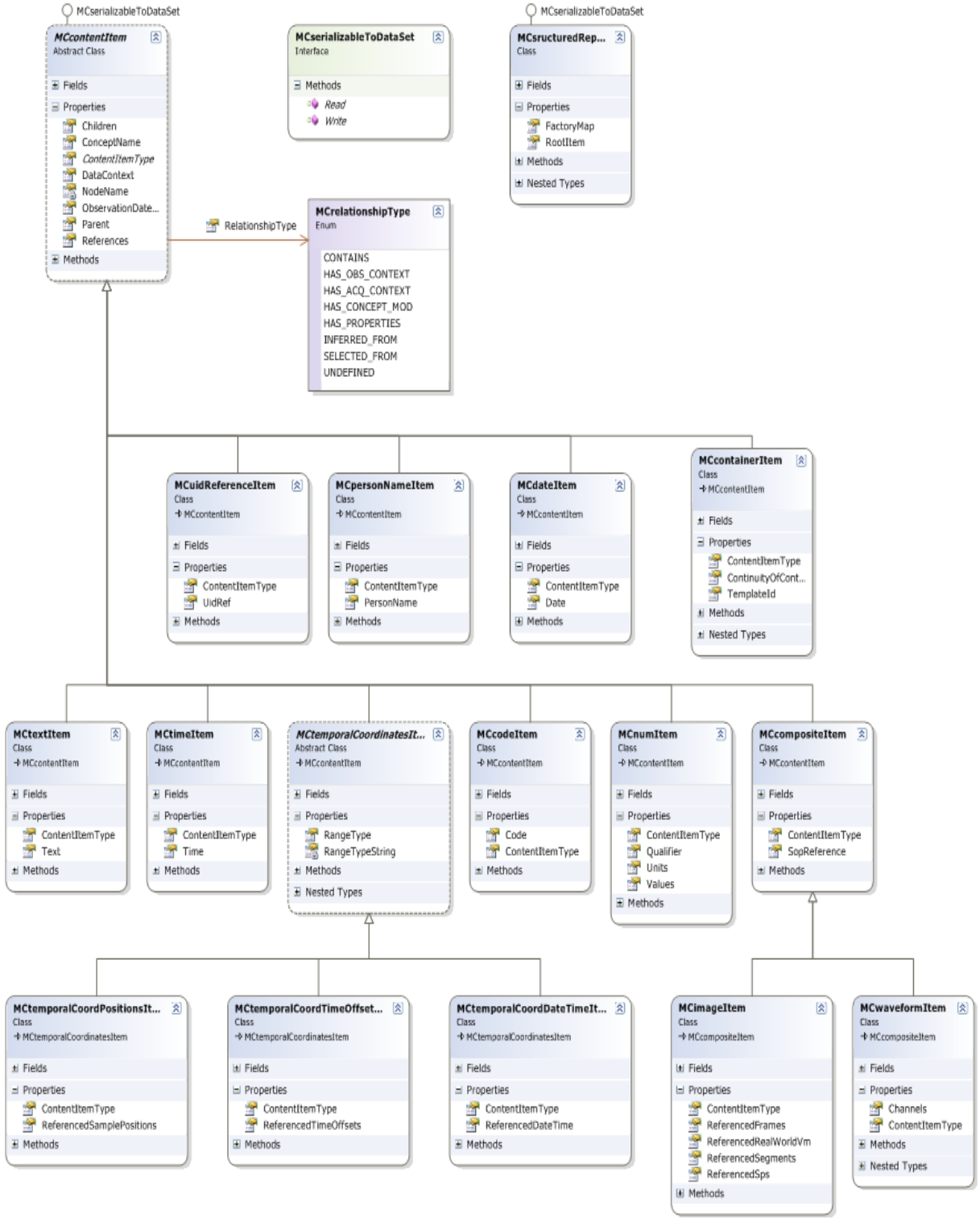

Figure 21: SR Class Diagram

In *Figure 21*, there are two top level classes.

- **MCstructuredReport** — This class is encapsulating the Structured Report Content Module and has utility functions for reading and writing content tree from DICOM datasets.
- **MCcontentItem** — This is a base class for all content item classes and encapsulates common functionality for all of them.

Each Content Item type is implemented as a separate class with the specific data exposed as public properties. All Content Item classes are implementing the ***MCserializableToDataSet*** interface that is used by the Toolkit to read and write individual Content Items.

Extending Toolkit Classes

The Merge DICOM Toolkit allows you to extend existing Content Item classes by providing your own derived classes from existing types. For example, if you want to store the Content Item data in your own format or save extra DICOM attributes that are not covered by the Toolkit classes. Once you created your own extended class, you need to register it with the class factory that is responsible for creating classes during reading data from DICOM datasets. By default, Toolkit will create a known class per each Content Item Type according to *Table 21*. The class factory is implemented in the **MCstructuredReport** class and can be updated by calling the following function:

```
public void UpdateItemFactory(ContentItemType itemType,
       Type classType)
```

Note: The class factory registration is not global and the registration shall be done per instance of the **MCstructuredReport** class.

Encoding SR Documents

The creation of the SR document involves following steps:

1. Creating a new **MCstructuredReport** object.
2. Adding Content Items (nodes) to the tree based on the templates definition.
3. Creating a new dataset.
4. Saving SR Content to the dataset.
5. Adding Patient/Study/Series and other attributes required by the IOD definition,
6. Saving the result dataset object to a file.

To create a new SR, you need to know the IOD type you are creating and the templates that will be used to generate the SR Document Content.

Key Object Selection Example

The Key Object Selection document is constrained by a single template. The following template is taken from the DICOM Part 16.

TID 2010
KEY OBJECT SELECTION
Type: Non-Extensible

	NL	Rel with Parent	VT	Concept Name	VM	Req Type	Condition	Value Set Constraint
1			CONTAINER	DCID(7010) Key Object Selection Document Titles	1	M		Root node
2	>	HAS CONCEPT MOD	CODE	EV (113011, DCM, "Document Title Modifier")	1-n	U		
3	>	HAS CONCEPT MOD	CODE	EV (113011, DCM, "Document Title Modifier")	1	UC	IF Row 1 Concept Name = (113001, DCM, "Rejected for Quality Reasons") or (113010, DCM," Quality Issue")	DCID (7011)
4	>	HAS CONCEPT MOD	CODE	EV (113011, DCM, "Document Title Modifier")	1	MC	IF Row 1 Concept Name = (113013, DCM, "Best In Set")	DCID (7012)
5	>	HAS CONCEPT MOD	INCLUDE	DTID(1204) Language of Content Item and Descendants	1	U		
6	>	HAS OBS CONTEXT	INCLUDE	DTID(1002) Observer Context	1-n	U		
7	>	CONTAINS	TEXT	EV(113012, DCM, "Key Object Description")	1	U		
8	>	CONTAINS	IMAGE	Purpose of Reference shall not be present	1-n	MC	At least one of Rows 8, 9 and 10 shall be present	
9	>	CONTAINS	WAVEFORM	Purpose of Reference shall not be present	1-n	MC	At least one of Rows 8, 9 and 10 shall be present	
10	>	CONTAINS	COMPOSITE	Purpose of Reference shall not be present	1-n	MC	At least one of Rows 8, 9 and 10 shall be present	

The code below generates a valid DICOM KO object and illustrates how the template is encoded using the Merge DICOM oolkit functions.

```
private MCdataSet CreateKO()
{
    MCcontentItem item;

    /*
    *  Create a KEY OBJECT DOCUMENT
    *  The template ID is 2010 and we used the "Best in Set"
    *  context ID from the CID 7010.
    */
    MCsructuredReport sr = new MCstructuredReport("2010",
                MCcontainerItem.Continuity.SEPARATE,
                   new MCbasicCodedEntry("113013", "DCM", "Best In Set"));

    /*
    *  Skipping Row 2 and 3 of the template and encoding Row 4.
    *  The code is taken from the CID 7012.
    */
    item = new MCcodeItem(new MCbasicCodedEntry("113015", "DCM",
"Series"),
             new MCbasicCodedEntry("113011", "DCM",
                                   "Document Title Modifier"));
    sr.RootItem.AddChild(item, MCrelationshipType.HAS_CONCEPT_MOD);

    /*
    * Skipping Row 5 and 6 of the template and encoding Row 7.
    * The code is taken from the CID 7012.
    * The text value shall describe the image selection.
    */
    item = new MCtextItem("Doctor's comments on selection",
               new MCbasicCodedEntry("113012", "DCM",
                                   "Key Object Description"));
    sr.RootItem.AddChild(item, MCrelationshipType.CONTAINS);

    /*
    * Adding an IMAGE from Row 8.
    * The values "1.2.3.4.1", "1.2.3.4.5.1" suppose to be an image SOP
Class
    * and SOP Instance.
    */
    item = new MCimageItem(new MCsopInstanceReference("1.2.3.4.1",
                                                      "1.2.3.4.5.1"));
    sr.RootItem.AddChild(item, MCrelationshipType.CONTAINS);

    /* Creating a new DataSet */
    MCdataSet dataSet = new MCdataSet(MCdimseService.C_STORE_RQ,
                                      "KEY_OBJECT_SELECTION_DOC");
    /* Saving SR Document contentent into the DataSet */
    sr.Write(dataSet);

    /*
    * Adding other root lvel attributes
    */
    dataSet.setValue(MCdicom.SOP_CLASS_UID, "1.2.840.10008.5.1.4.1.1.88.59");
    dataSet.setValue(MCdicom.SOP_INSTANCE_UID, "1.2.3.4.5.6.7.300");
    dataSet.setValue(MCdicom.STUDY_DATE, "19991029");
    dataSet.setValue(MCdicom.CONTENT_DATE, "19991029");
    dataSet.setValue(MCdicom.STUDY_TIME, "154500");
    dataSet.setValue(MCdicom.CONTENT_TIME, "154510");
    dataSet.setValue(MCdicom.ACCESSION_NUMBER, "123456");
    dataSet.setValue(MCdicom.MODALITY, "KO");
    dataSet.setValue(MCdicom.MANUFACTURER, "MERGE");
    dataSet.setValue(MCdicom.REFERRING_PHYSICIANS_NAME,
                                      "Luke^Will^^Dr.^M.D.");
    dataSet.setValue(MCdicom.REFERENCED_PERFORMED_PROCEDURE_STEP_SEQUENCE,
                                      null);
    dataSet.setValue(MCdicom.PATIENTS_NAME, "Jane^Doo");
    dataSet.setValue(MCdicom.PATIENT_ID, "234567");
    dataSet.setValue(MCdicom.PATIENTS_BIRTH_DATE, "19991109");
    dataSet.setValue(MCdicom.PATIENTS_SEX, "F");
```

```
    dataSet.setValue(MCdicom.STUDY_INSTANCE_UID, "1.2.3.4.5.6.7.100");
    dataSet.setValue(MCdicom.SERIES_INSTANCE_UID, "1.2.3.4.5.6.7.200");
    dataSet.setValue(MCdicom.STUDY_ID, "345678");
    dataSet.setValue(MCdicom.SERIES_NUMBER, "1");
    dataSet.setValue(MCdicom.INSTANCE_NUMBER, "1");
    dataSet.setValue(MCdicom.PERFORMED_PROCEDURE_CODE_SEQUENCE, null);
    MCitem item1 = new MCitem("HIERARCHICAL_SOP_INST_REF_MACRO");
    item1.setValue(MCdicom.STUDY_INSTANCE_UID, "1.2.3.4.5.6.7.100");
    MCitem item2 = new MCitem("HIERARCHICAL_SERIES_REF_MACRO");
    item2.setValue(MCdicom.SERIES_INSTANCE_UID, "1.2.3.4.5.6.7.200");
    MCitem item3 = new MCitem("REF_SOP");
    /** following UIDs are the same as used in the Row 8 item ****/
    item3.setValue(MCdicom.REFERENCED_SOP_CLASS_UID, "1.2.3.4.1");
    item3.setValue(MCdicom.REFERENCED_SOP_INSTANCE_UID, "1.2.3.4.5.1");
    item2.setValue(MCdicom.REFERENCED_SOP_SEQUENCE, item3);
    item1.setValue(MCdicom.REFERENCED_SERIES_SEQUENCE, item2);
    dataSet.setValue(MCdicom.CURRENT_REQUESTED_PROCEDURE_EVIDENCE_SEQUENCE,
                                    item1);

    return dataSet;
}
```

Reading SR Documents

Reading SR Documents is done in a similar way to encoding, but in reverse sequence.

1. Reading a File or receiving a message object.
2. Reading root level attributes.
3. Creating a new **MCstructuredReport** object.
4. Reading SR Content from the dataset.
5. Traversing SR content tree and accessing Content Node attributes.

The following code illustrates a reading sequence for the Key Object Document generated above.

```
private void ReadKO(MCdataSet dataset)
{
    // Create a new SR instance and fill it from the dataset
    MCsructuredReport sr = new MCstructuredReport();
    sr.Read(dataset);
    // Print information from the root CONTAINER item
    Console.WriteLine("Document Title: " +
                        sr.RootItem.ConceptName.CodeMeaning);
    Console.WriteLine("Template Id: " + sr.RootItem.TemplateId);

    // Reading the first children as a CODE item
    if (sr.RootItem.Children[0] is MCcodeItem)
    {
        MCcodeItem codeItem = sr.RootItem.Children[0] as MCcodeItem;
        Console.Write(codeItem.ContentItemType + ": ");
        Console.WriteLine(codeItem.ConceptName.CodeMeaning + ": " +
                            codeItem.Code.CodeMeaning);
    }
    // Reading the next children as a TEXT item
    if (sr.RootItem.Children[1] is MCtextItem)
    {
        MCtextItem textItem = sr.RootItem.Children[1] as MCtextItem;
        Console.Write(textItem.ContentItemType + ": ");
        Console.WriteLine(textItem.ConceptName.CodeMeaning + ": " +
                        textItem.Text);
    }
```

```
// Reading the next children as a IMAGE item
if (sr.RootItem.Children[2] is MCimageItem)
{
    MCimageItem imageItem = sr.RootItem.Children[2] as MCimageItem;
    Console.Write(imageItem.ContentItemType + ": ");
    Console.WriteLine("Sop Class: " +
          imageItem.SopReference.ReferencedSopClassUid +
          "  Sop Instance: " +
          imageItem.SopReference.ReferencedSopInstanceUid);
}
```

Deploying Applications

There are several issues to consider when deploying a Merge DICOM based application. These include deciding which Merge DICOM files are needed for your application, how to set important configuration options to reduce problems in the field, and how to deal with potential UN VR problems. The following sections describe these issues in further detail.

Merge DICOM Required Files

There are a number of files required by Merge DICOM applications. These files are described in *Table 25*.

Table 25: Files needed when deploying an application.

File	Description and Use
Mergecom.dll	.NET Merge DICOM library wrapper. This library services your calls to the Native Merge DICOM Toolkit Library.
Mergecom.Native.dll	Native Merge DICOM Toolkit library. (required for deployments on 32-bit platforms)
Mergecom.Native64.dll	Native Merge DICOM Toolkit library for 64-bit processes. (required for deployments on 64-bit platforms)
Picn20.dll Picn6220.dll Picn6320.dll Picn6420.dll Picn6520.dll Picn6820.dll Picn6920.dll	Pegasus libraries used for compression. Pegasus Imaging Corporation (Hwww.jpg.comH) (required for deployments on 32-bit platforms)

File	Description and Use
picx20.dll picx6220.ssm picx6320.ssm picx6420.ssm picx6520.ssm picx6820.ssm picx6920.ssm	64-bit Pegasus libraries used for compression. Pegasus Imaging Corporation (Hwww.jpg.comH) (required for deployments on 64-bit platforms)
merge.ini	Merge DICOM initialization file. This file contains logging configuration and path names for the other configuration files.
mergecom.pro	Merge DICOM system profile. This file contains general run-time configuration options.
mergecom.app	Merge DICOM application profile. This file contains configuration information about the services supported by the Merge DICOM application and information about remote DICOM applications.
mergecom.srv	Merge DICOM services file. This file contains information about the services supported by Merge DICOM Toolkit.
mrgcom3.msg	Merge DICOM message information file. This file contains validation information for DICOM messages. This file is required if a non-empty MCdataSet, MCitem, MCdimseMessage or MCfile object is constructed by the application; or if any validate or validateAttribute is called; or if an MCdir object is constructed.
mrgcom3.dct	Merge DICOM data dictionary file.

Configuration Options

The majority of Merge DICOM Toolkit's configuration options can be used to solve interoperability problems in the field. There are some options, however, that can be set before deploying a Merge DICOM application to help reduce potential problems. These options are listed in *Table 26* with descriptions of how they can be set.

Table 26: Configuration options to consider when deploying an application.

Configuration Option	Description
ACCEPT_ANY_APPLICATION_TITLE	When set to NO, Merge DICOM requires that the Application Entity title sent in an association request match one of the registered application titles for the SCP. When there is no match, the association will be automatically rejected. Setting this option to YES will eliminate some association negotiation problems in the field for SCP applications.
ACCEPT_ANY_HOSTNAME	When set to NO, Merge DICOM will attempt to resolve the IP address of the SCU application into a hostname. If this resolution cannot be done, the association will automatically be rejected. Setting this option to YES will reduce configuration problems in the field for SCP applications.
EXPORT_UN_VR_TO_MEDIA	Setting this option to NO will cause UN VR attributes to not be exported when writing DICOM Part 10 format files with the writeFile or writeFileByCallback methods of the MCmediaStorageService class. See the following sections for a further discussion of UN VR.
EXPORT_UN_VR_TO_NETWORK	Setting this option to NO will cause UN VR attributes to not be exported over the network when sending messages using the MCdimseService class. See the following sections for a further discussion of UN VR.

Configuration Option	Description
IMPLEMENTATION_CLASS_UID	The Implementation Class UID is used to identify in a unique manner a specific class of implementation. PS3.7 of DICOM states: "(The Implementation Class UID) is intended to provide respective (each network node knows the other's implementation identity) and non-ambiguous identification in the event of communication problems encountered between two nodes." PS3.7 of DICOM further defines how this UID should be defined: "different equipment of the same type or product line (but having different serial numbers) shall use the same Implementation Class UID if they share the same implementation environment (i.e., software)."
IMPLEMENTATION_VERSION	The Implementation Version is intended to distinguish between software versions of an implementation. It should be set to the version of the Merge DICOM application.

UN VR

What to do when all network partners are not aware of the UN VR

DICOM Supplement 14, Unknown Value Representation, became a part of the DICOM standard on June 3, 1997. This supplement added a new value representation, UN, to the DICOM standard. It was developed to fix two related holes in the DICOM standard:

When standard or private attributes were received in an implicit value representation (VR) transfer syntax, and the user does not have a knowledge of the VR of the attributes, there is no way to represent the VR for these attributes in an explicit VR transfer syntax.

Every time a new VR is added to the standard, there is no way to determine if the length field in explicit value representation transfer syntaxes should be encoded as 2 bytes or 4 bytes, so a general parser could not be properly written to handle future VRs.

The need for this supplement is mainly for use in "archive" systems. An "archive" will typically want to preserve the private attributes contained within a message for later use. There also may be a need to add support for new image objects with new VRs to an "archive" system without having to change the software.

Unfortunately, the method that Supplement 14 specifies for encoding UN value representation attributes is in some cases not compatible with older DICOM implementations. Versions previous to 2.2.2 of the Merge DICOM toolkit do not parse these attributes properly. The MCassociation `read` method will fail and the association will be aborted if a UN VR attribute is received. This has obviously caused a variety of interoperability problems in the field.

The typical DICOM scenario where UN VR can cause a DICOM communication failure is the following: a modality exports a series of images to a PACS or "archive" system via the DICOM storage service class. The images were encoded in the implicit VR little endian transfer syntax and contain multiple

private attributes. Later, a DICOM workstation decided to retrieve the images from the "archive" or PACS system. The workstation does not yet support UN VR, however, the PACS or "archive" system does. The workstation uses the DICOM query/retrieve service class to retrieve the series of images. When the images are exported to the workstation with an explicit VR transfer syntax, the workstation fails to parse the first image received when it encounters the first UN VR attribute, and the association is automatically aborted by the workstation.

We have added several methods to solve this interoperability problem through the Merge DICOM toolkit's configuration files. For SCU systems that are exporting UN VR tags to systems that cannot handle them, the following can be done:

SCU application strategy

Configure the SCU to only use the Implicit VR Little Endian transfer syntax when exporting objects. This can be done through the use of transfer syntax lists within the `mergecom.app` file or through commenting out the UID definitions for the other transfer syntaxes within the mergecom.pro file.

Set the **`UNKNOWN_VR_CODE`** configuration option in the mergecom.pro file to 'OB'. This forces unknown VR attributes to be encoded as OB instead of as UN. All implementations can handle OB encoding. There are several drawbacks to this option. If the attributes are encoded as OB, it is harder for these attributes to be converted back to their normal VR. Secondly, this option changes all instances of the UN VR into OB. Systems that can handle the UN VR will now also receive these attributes as OB.

Set the **`EXPORT_UN_VR_TO_NETWORK`** configuration option to 'No'. This will cause the Merge DICOM toolkit to not export attributes encoded as UN VR to the network.

SCP application strategy

For SCP systems receiving UN VR tags when they cannot handle them, the following can be done:

Configure the SCP to only negotiate the Implicit VR Little Endian transfer syntax when receiving objects.

With the help of these options, most UN VR problems in the field can be fixed simply by changing configuration values with the Merge DICOM toolkit.

Appendix A: Frequently Asked Questions

This appendix lists some frequently asked questions by toolkit users.

1. *It is inconvenient to set absolute paths for the various configuration options in the merge.ini and mergecom.pro files that need them. Is there a way to make these pathnames be configurable at run-time?*

   Merge DICOM allows the placement of environment variables in these pathnames. This allows setting of a root directory for these pathnames. The following is an example of how this functionality is used in our configuration files:

   ```
   MERGECOM_PRO = $(MERGE_ROOT)\mc3apps\mergecom.pro
   ```

   In this example, `MERGE_ROOT` would be an environment variable.

   A special macro "MC3INIDIR" is used to represent the directory where "merge.ini" is. It is used like the environment variable with the difference that it is automatically resolved and does not need to be set.

   If MERGECOM_3_PROFILE, MERGECOM_3_SERVICES or MERGECOM_3_APPLICATIONS contain relative paths with a prefix "$(MC3INIDIR)" or "%MC3INIDIR%", the toolkit considers the path relative to the location of the "merge.ini" file.

   For example:

   ```
   MERGECOM_3_PROFILE = $(MC3INIDIR)../config/mergecom.pro
   ```

   The path of the profile file is "../config/mergecom.pro" relative to the location of the "merge.ini" file.

2. *I am testing the sample applications for the first time and cannot get the client (SCU) application to connect to the server (SCP) for any of the sample applications. The MCapplication **requestAssociation** method is throwing an exception. It appears as though the connection is opening, but it is quickly dropped. Why is this happening?*

   As a security measure, the MCassociation **startListening** method used in SCPs attempts to determine the hostname of SCUs connecting to it. If it cannot determine the remote hostname, it will drop the connection. The startListening method uses the local system's host file or its configured domain name server to translate the SCU's IP address into its hostname. By configuring the SCU's hostname in your local hosts file, this problem will be eliminated. Also, the **`ACCEPT_ANY_HOSTNAME`** configuration value in the mergecom.pro file disables this checking.

3. *What can be done to reduce the memory requirements of the Merge DICOM Toolkit?*

   Performance Tuning

   There are several methods for reducing the memory requirements of Merge DICOM Toolkit. The first approach is to construct "empty" MCfile, MCdataSet and MCdimseMessage objects by not specifying any service or command parameters. These constructors reduce memory by not reading in all of the information needed for validation of messages and files respectively. This approach will also improve performance.

There are several configuration values that reduce Merge DICOM Toolkit's memory requirements. The following describes each of these options:

FORCE_OPEN_EMPTY_ITEM — This configuration option performs the same function as constructing empty MCdimseMessage objects, except that it is for items. It is especially useful for reducing the amount of memory used when creating large DICOMDIRs.

LARGE_DATA_STORE and **LARGE_DATA_SIZE** — These options control the ability of Merge DICOM to store pixel data in temporary files instead of RAM. This functionality is enabled by setting **LARGE_DATA_STORE** to **FILE**, and adjusting **LARGE_DATA_SIZE** to the size of data element that you want spooled to temporary file. Note however that this will decrease performance.

4. *What can be done to increase the performance of the Merge DICOM Toolkit?*

   There are several Merge DICOM configuration values that impact performance in different ways. The following is a summary of these options:

   ELIMINATE_ITEM_REFERENCES — This option improves the performance of removeMessageValues mthod in MCdimseMessage, clear method in MCattributeSet and removeFileValues method in MCfile. This option will disable functionality within the toolkit that causes the toolkit to search all currently open message objects for references to an item that is being freed by one of these calls. This call is especially useful when your application uses very large DICOMDIR files.

   PDU_MAXIMUM_LENGTH — This option sets the maximum sized PDU that the toolkit will receive. If during association negotiation the maximum sized PDU of the system negotiating with the toolkit application is larger than this value, the PDU size will be limited to this value. Increasing this value increases the amount of data that is passed to the TCP/IP level. This may increase network performance of the library.

   WORK_BUFFER_SIZE — This option specifies how the toolkit buffers data before storing it or passing it to a user's callback class. Setting higher values for this option will increase performance.

   TCPIP_RECEIVE_BUFFER_SIZE — This option sets the TCP/IP receive buffer size. Higher values for this buffer generally will increase the network performance of the toolkit for server (SCP) applications. This value should also be slightly larger than the **PDU_MAXIMUM_LENGTH** to increase performance. Setting this value to an even multiple of the MSS (1460 bytes) will help increase performance on most platforms.

   TCPIP_SEND_BUFFER_SIZE — This option sets the TCP/IP send buffer size. Higher values for this buffer generally will increase the network performance of the toolkit for client (SCU) applications. This value should also be slightly larger than the **PDU_MAXIMUM_LENGTH** to increase performance. Setting this value to an even multiple of the MSS (1460 bytes) will help increase performance on most platforms

5. *Which of the options listed above have the greatest impact on network performance?*

   The **TCPIP_RECEIVE_BUFFER_SIZE** and **TCPIP_SEND_BUFFER_SIZE** configuration options have the greatest impact on network performance. Setting these properly directly increases the network performance of Merge DICOM Toolkit.

6. *I am sending 8-bit images with Merge DICOM Toolkit, however, after sending the data to another system, the pixel data is byte swapped incorrectly. What is causing this problem?*

   The Merge DICOM Toolkit Users Manual contains the section "8-bit Pixel Data" (page 99) which describes this problem. This is typically only a problem on Big Endian machines. To summarize the problem, on big endian machines, we expect 8-bit data to be byte swapped. We do not look at the "bits allocated" and "bits stored" tags to determine that the pixel data itself is 8-bit data, we always treat pixel data `(7fe0,0010)` as OW. The pixel data must be assigned as byte swapped.

7. *I recently upgraded to a new release of the Merge DICOM Toolkit. Since this upgrade, exceptions are being thrown by the MCattributeSet attribute encoding methods. This code worked before the upgrade. What is causing these problems?*

   The Merge DICOM data dictionary changes from release to release. In some cases, the identification number for a particular message type changes. When upgrading, if you do not change all of the data dictionary files, this error will occur. The following files should be upgraded with each release:

   ```
   mergecom.srv

   mrgcom3.msg

   mrgcom3.dct
   ```

Appendix B: Unique Identifiers (UIDs)

UIDs provide the capability to identify many different types of items. The purpose of UIDs are to guarantee the uniqueness of the these different types of items. DICOM uses UIDs to uniquely identify items such as SOP classes, image instances and network negotiation parameters. Part 5, Section 9 along with Annexes B and C of the DICOM Standard discusses how UIDs are composed, encoded and registered.

Summary of UID Composition

A UID is composed of a number of numeric values as defined by ISO 8824. The following is a typical example of a UID:

```
1.2.840.10008.2.45.1.12345
```

A UID is composed of two parts: a <root> and a <suffix> and has the following form:

```
UID = <root>.<suffix>
```

where <root> is assigned by a registration authority (e.g., ANSI) with the distinguishing component being the organization ID. The <root> portion of the UID uniquely identifies an organization while the <suffix> portion is used to uniquely identify a specific object within the scope of the organization. While the <root> component of the UID stays constant, the <suffix> portion will change in a manner that will provide uniqueness for objects that need UIDs.

Note: This implies that the organization is responsible for maintaining the uniqueness of the <suffix>.

For example, using the UID above, <root> = 1.2.840.10008 and <suffix> = 2.45.1.12345. Where the organization ID portion of the <root> (10008) distinguishes organizations from each other.

Note: The above example is typical for UIDs obtained by ANSI during the time when the DICOM standard was first released. The organization ID of 10008 has actually been assigned to NEMA and is used as part of the <root> for DICOM standard UIDs such as SOP Classes, Transfer Syntaxes, etc. For example, vendors creating images need to obtain their own organization ID and cannot use 10008.

For future UIDs, ISO has developed a joint relationship with CCITT and has changed the <root> structure. Therefore, new UIDs from ANSI will no longer be of the form 1.2.840.xxxxx. but are currently assigned using the form, <root> = 2.16.840.1.10008. Where, of course, 10008 is the organization ID.

Obtaining a UID

The <root> portion of the UID should be registered by an organization that guarantees global uniqueness. The American National Standards Institute (ANSI) is the registration authority for the United States. Other national registration authorities exist for nations throughout the world such as IBN in Belgium, AFNOR in France, BSI in Great Britain, DIN in Germany, and COSIRA in Canada.

Obtaining a UID From ANSI

ANSI is the registration authority for the US for organization names (i.e., <root>) under the global registration process established by the International Standards Organization (ISO) and the International Telegraph and Telephone Consultative Committee (CCITT). ANSI's registration service conforms with CCITT X.660 and ISO/IEC 9834-1. The ANSI organization name registration service assigns one name component to the hierarchy defined by CCITT and ISO/IEC.

An organization seeking registration may do so by submitting a Request for Registration application form along with a fee (as of August 1996 the fee is $1,000) to the Registration Coordinator. The Request for Registration application form can be obtained from ANSI by use of the following information:

American National Standards Institute

11 West 42nd Street

New York, New York 10036

TEL: 212.642.4900

FAX: 212.398.0023

Appendix C: Writing a DICOM Conformance Statement

Detailed below is a guideline for writing a DICOM conformance statement for your application. Since the Toolkit is *not* an application, this section only gives an outline of the DICOM services it supports. Responsibility for full DICOM conformance to particular SOP classes rests with the application developer, since many of the requirements for such conformance lie outside the realm of the Toolkit. For example, the high level behavior of Query/Retrieve service class SCUs and SCPs as defined in Part 4 of the DICOM standard, is implemented by the application developer in conjunction with the toolkit functionality.

Conformance Statement Sections

Implementation Model

The Implementation model consists of three sections:

- the Application Data Flow Diagram which specifyies the relationship between the Application Entities and the "external world" or Real-World activities;
- a functional description of each Application Entity; and
- the sequencing constraints among them.

Application Data Flow

As part of the Implementation model, an Application Data Flow Diagram is included. This diagram represents all of the Application Entities present in an implementation, and graphically depicts the relationship of the AE's use of DICOM to Real-World Activities as well as any applicable user interaction.

The Merge DICOM Toolkit provides the core functionality required to facilitate data flow between SCUs and SCPs.

Application conformance statements include a data flow diagram. An example is shown below for a simple Storage Service Class SCP.

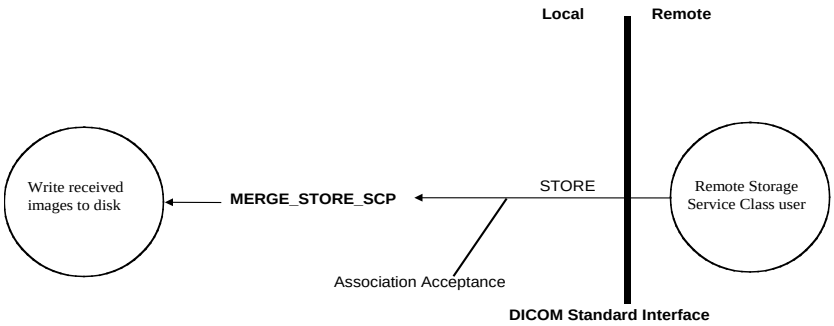

Figure C-1: MERGE_STORE_SCP application data flow diagram

Functional Definition of Application Entities (AE)

This section contains a functional definition for each individual, local Application Entity. It describes in general terms, the functions that are performed by the AE, and the DICOM services used to accomplish these functions. In this sense, "DICOM services" refers not only to DICOM Service Classes, but also to lower level DICOM services, such as Association Services.

Application conformance statements are described in this section with a general specification of functions to be performed by SCU or SCP.

Sequencing of Real World Activities

If applicable, this section will contain a description of sequencing as well as potential constraints on real-world activities. These include any applicable user interaction as performed by all the AEs. A UML sequence diagram that depicts the real-world activities as vertical bars, and shows events exchanged between them as arrows, is strongly recommended.

Application conformance statements are included in this section along with any associated sequence of real-world activities. For example, a Storage Service Class SCP might perform the following real-world activities: store an image, modify it in some defined manner, act as a Storage Service Class SCU and forward the modified image somewhere.

AE Specifications

The next section in the DICOM Conformance Statement is a set of Application Entity specifications. There is one specification for the AE. Each individual AE specification has a subsection. There are as many of these subsections as there are different AE's in the implementation. That is, if there are two distinct AEs, then there are two subsections. The Merge DICOM Toolkit uses the mergecom.app configuration file to read configuration parameters for each AE. The following subsections are filled in for each AE:

- Application Entity
  - SOP Classes
  - Association Policies
    - General
    - Number of Associations
    - Asynchronous Nature
    - Implementation Identifying Information
  - Association Initiation Policy
    - Activity
      - Description and Sequencing of Activities
      - Proposed Presentation Contexts
      - SOP Specific Conformance for SOP Class(es)
  - Association Acceptance Policy
    - Activity
      - Description and sequencing of Activities
      - Accepted Presentation Contexts
      - SOP Specific Conformance for SOP Class(es)

SOP Classes

Application conformance statements specify the DICOM SOPs which are supported by each Application Entity. For SCP Entities, the initiation of associations. Please see the "System Profile" section in ANNEX B: CONFIGURATION PARAMETERS and the "MC_Wait_For_Association" or "MC_Wait_For_Secure_Association" definition in this Reference Manual. For SCU Entities, the list of supported SOP classes will correspond to the services specified in "mergecom.app" for any SCPs to which the SCU wishes to connect.

Number of Associations

The Merge DICOM Toolkit does not impose any limit on the number of simultaneous associations that can be requested or accepted. The only limitation on the number of simultaneous associations is imposed by the operating system and available resources. However, if your application enforces this limit, it is defined here.

The MAX_PENDING_CONNECTIONSsetting in the "mergecom.pro" file refers to the maximum number of outstanding connection requests per listener socket. It does not limit the maximum number of simultaneous associations.

Asynchronous Nature

Merge DICOM Toolkit does not currently support multiple outstanding transactions over a single association.

Implementation Identifying Information

Application conformance statements specify the Implementation Class Unique Identifier (UID) for the application, as well as the Implementation version name. These identifiers are taken from the mergecom.pro configuration file under the following keys:

```
IMPLEMENTATION_CLASS_UID

IMPLEMENTATION_VERSION
```

This UID must follow the syntax rules specified in Part 5 of the DICOM standard.

Proposed or Accepted Presentation Contexts

Application conformance statements specify all presentation contexts that are used for association negotiation. A presentation context consists of:

- an Abstract Syntax which is a DICOM service class name and unique identifier(UID);
- a transfer syntax name and UID. A transfer syntax represents a set of data encoding rules that are specified in the "mergecom.pro" file. Please see the "System Profile" section in Appendix B: Configuration Parameters;
- the role that the application will perform within the service class. The roles associated with a particular service class are discussed in Part 4 of the DICOM standard;

- any extended negotiation information used when creating associations. See the "MC_Get_Negotiation_Info" function in the Merge DICOM Reference Manual; and; and
- any rules that govern the acceptance of presentation contexts for the AE. This includes rules for which combinations of Abstract/Transfer Syntaxes are acceptable, and rules for prioritization of presentation contexts. Rules that govern selection of transfer syntax within a presentation context are stated here. Please see the "Application Profile" section in the Appendix B: Configuration Parameters. Also, see the "MC_Get_Association_Info" function in the Merge DICOM Reference Manual to learn about the presentation contexts that are queryable by an application program.

Refer to *Table 27* for an example.

Table 27: Example Presentation Context

Presentation Context Table					
Abstract Syntax		**Transfer Syntax**		**Role**	**Extended**
Name	**UID**	**Name List**	**UID List**		**Negotiation**
Computed Radiography Image Storage	1.2.840.10008.5.1.4.1.1.1	DICOM Implicit VR Little Endian	1.2.840.10008.1.2	SCP	None
		DICOM Explicit VR Little Endian	1.2.840.10008.1.2.1		
		DICOM Explicit VR Big Endian	1.2.840.10008.1.2.2		

Merge DICOM Toolkit uses mergecom.app configuration settings to specify presentation contexts shown above.

SOP Specific Conformance

This section includes the SOP specific behavior, i.e., error codes, error and exception handling and time-outs, etc. The information is described in the SOP specific Conformance Statement section of PS 3.4 (or relevant private SOP definition).

Transfer Syntax Selection Policies

Merge DICOM Toolkit uses the following policy when selecting a transfer syntax:

- An SCU offers any transfer syntaxes which are defined in it's mergecom.pro file.

- The SCP prefers it's native byte ordering, and will prefer explicit over implicit VR.

Network Interfaces

Physical Network Interface

Merge DICOM Toolkit runs over the TCP/IP protocol stack on any physical interconnection media supporting the TCP/IP stack.

IPv4 and IPv6 Support

Merge DICOM Toolkit supports both IPv4 and IPV6 protocols and is configurable in the system profile.

Configuration

Refer to the Appendix B: Configuration Parameters for complete configuration information.

Applications reference four (4) configuration files. The first, merge.ini, is found through the MERGE_INI environment variable. They are as follows:

- merge.ini — Specifies the names of the other three (3) configuration files and also contains message logging parameters.
- mergecom.pro — Specifies run-time parameters for the application.
- mergecom.app — Defines service lists and applications on other network nodes to which connections are possible.
- mergecom.srv — Service and sequence definitions.

AE Title/Presentation Address Mapping

Presentation address mapping is configured in the mergecom.app file. The Presentation Address of an SCU/SCP application is specified by configuring the Listen Port in the mergecom.pro file, and specifying the AE title for the SCU/SCP within the application itself.

Configurable Parameters

The mergecom.pro configuration file can be used to set or modify other lower-level communication parameters. This includes time-outs and other parameters. Some information about supported SOP classes is also stored here. **Most parameters in this file should NEVER be changed. Doing so may compromise DICOM conformance.** Before modifying any parameters, such as time-out, be sure to have a backup of the originally supplied mergecom.pro file. Also, before modifying other parameters, you should consider contacting Merge Healthcare for advice.

PDU size

The maximum PDU size is configurable with a minimum of 4,096 bytes.

Application conformance statements specify the chosen PDU (Protocol Data Units) size and any general rules governing the initiation of associations. Please see the "System Profile" section of the Merge DICOM Reference Manual for further information about configuring the PDU size.

Extensions/Specializations/Privatizations

Standard Extended/Specialized/Private SOPs

Application conformance statements list extended, specialized, or private SOPs that are supported.

Private Transfer Syntaxes

This section describes private transfer syntaxes that are listed in the Transfer Syntax Tables. See the System Profile section in Appendix B: Configuration Parameters for details.

Appendix D: Configuration Parameters

This appendix describes each configuration parameter in detail. Information contained in these tables is the parameter names, descriptions and sections where it is contained. The parameters are listed alphabetically and organized by the initialization file where they are used.

Initialization File

The following parameters are recognized by Merge DICOM in the initialization file.

Table 28: Initialization file parameters

Name	Section	Description
BLANK_FILL_LOG_FILE	MergeCOM3	This parameter informs the toolkit whether or not to expand the log file to its maximum size on initialization. Setting this value to "NO" will decrease the time spent in the MC.mcInitialization call but increase the time spent doing actual logging while the application is running. DEFAULT: YES
ERROR_MESSAGE	MergeCOM3	This parameter informs the toolkit to log error messages.
INFO_MESSAGE	MergeCOM3	This parameter informs the toolkit to log error messages.
LOG_FILE	MergeCOM3	This is the name of the Merge DICOM message log. The file will be [re-]created by Merge DICOM Toolkit. This parameter is ignored by embedded toolkits. DEFAULT: ./merge.log
LOG_FILE_BACKUP	MergeCOM3	This is a Boolean parameter that tells Merge DICOM to create a backup of the log file before starting a new log. If "ON", any existing log file is renamed with a file extension of .L*nn* where *nn* is an integer number between 01 and 99. DEFAULT: OFF.

Name	Section	Description
LOG_FILE_LINE_LENGTH	MergeCOM3	This option specifies the number of characters that occur on a line within the merge.log file. DEFAULT: 78 MINIMUM: 16 MAXIMUM: 254
LOG_FILE_SIZE	MergeCOM3	This is the number of 80-byte records which will be created for the log file, i.e. the number of 80 character lines in the log file. If BLANK_FILL_LOG_FILE is set to YES, the file is initialized to all binary zeros before the first message is logged. DEFAULT: 1000
LOG_MEMORY_SIZE	MergeCOM3	This is the number of 80-byte records which will be created for the memory log, i.e. the number of 80 character lines in the memory file. Note that this option is ignored when using the .NET Assembly. DEFAULT: 1024.
MERGECOM_3_APPLICATIONS	MergeCOM3	File containing the Merge DICOM application configurations
MERGECOM_3_PROFILE	MergeCOM3	File containing the Merge DICOM system profile parameters
MERGECOM_3_SERVICES	MergeCOM3	File containing the Merge DICOM system service and message definitions
NUM_HISTORICAL_LOG_FILES	MergeCOM3	This parameter informs the toolkit of the number of historical log files to keep. The valid range of number for this parameter is 1 - 99. The historical log files are named *basename*.L01 to *basename*.L*XX* where *basename*.L*XX* is the latest log file. The *basename* is determined by the LOG_FILE parameter. When the maximum number of historical log files is met, the oldest log file is deleted and the log files are renamed. Note that a new log file is created each time the library is initialized. This parameter is only used when LOG_FILE_BACKUP is set to YES.

Name	Section	Description
T3_MESSAGE	MergeCOM3	This logging level parameter informs the toolkit to log messages relating to association negotiation.
T4_MESSAGE	MergeCOM3	This logging level parameter informs the toolkit to log messages when incoming associations are automatically rejected.
T5_MESSAGE	MergeCOM3	This logging level parameter informs the toolkit to log messages relating to regular and extended validation.
T6_MESSAGE	MergeCOM3	This logging level parameter informs the toolkit to log messages relating to configuration.
T7_MESSAGE	MergeCOM3	This logging level parameter informs the toolkit to log messages relating to logging of command level attributes in messages sent or received.
T8_MESSAGE	MergeCOM3	This logging level parameter informs the toolkit to log messages relating to the streaming in and out of messages and file objects.
T9_MESSAGE	MergeCOM3	This logging level parameter informs the toolkit to log messages relating to PDU's sent and received. NOTE: Receipt and transmission of P-DATA PDU's are logged; not the actual PDU itself.
WARNING_MESSAGE	MergeCOM3	This parameter informs the toolkit to log warning messages.

Application Profile

The application profile is a configuration file that is application dependent. The application profile does not set specific parameters. It sets parameters related to characteristics of your own application entity. A detailed description of the application profile can be found in *Merge DICOM Toolkit: Users Manual*.

This section will define how each parameter should be defined within the application profile.

Sections

The application profile contains the following sections.

Table 29: Application profile section headings

Section	Description
<remote_application_title>	Section describing a remote DICOM Application Entity title(s). The remote Application Entity titles listed here must be 1 to 16 bytes in length with no embedded spaces. Simply, this section is where you list the DICOM applications you want to communicate with.
<service_list_name>	List(s) of DICOM services that will be provided by the Application Entities listed in the [<remote_application_title>] sections. The service names listed here must be 1 to 33 bytes in length with no embedded spaces. Simply, this section is where you list the services that are provided by the remote DICOM applications.
<syntax_list_name>	List(s) of DICOM transfer syntaxes that will be supported by the services listed in the [<service_list_name>] sections. The transfer syntaxes must be one of those listed in *Table 34*.

Parameters

The application profile contains the following parameters:

Table 30: Application profile section headers

Parameter	Section	Description
PORT_NUMBER	<remote_application_title>	This parameter is the TCP/IP port on which the remote DICOM system listens for connections. The commonly used port number is 104. This default value may be overridden by the requestAssociation method of the MCassociation class.
HOST_NAME	<remote_application_title>	This parameter is the name of the remote host as it is known to your TCP/IP system. This default value may be overridden by the requestAssociation method of the MCassociation class. The parameters value must be 1 to 19 bytes in length with no embedded spaces. NOTE that a numeric internet address may be used: e.g., 192.204.32.1

Parameter	Section	Description
SERVICE_LIST	<remote_application_title>	This parameter is the name of a section in the application profile which provides a list of services for which local applications will negotiate when attempting to establish an association. This is a default list; another list may be specified in the requestAssociation method of the MCassociation class. The parameters value names must be 1 to 33 bytes in length with no embedded spaces.

The `SERVICE_LIST` section of the Application Profile is used to describe the DICOM services that will be negotiated by the listed Application Entity. The parameter values are text strings recognizable by the Merge DICOM toolkit. These strings are defined in detail in `message.txt`. This file is located in the `mc3msg` directory of your distribution. The following is a list of currently supported services:

Table 31: Application profile parameters

Merge DICOM Service Parameter	DICOM Service Class
ARTERIAL_PULSE_WAVEFORM	Storage
AUTOREFRACTION_MEASUREMENTS	Storage
BASIC_ANNOTATION_BOX	Print Management
BASIC_COLOR_IMAGE_BOX	Print Management
BASIC_FILM_BOX	Print Management
BASIC_FILM_SESSION	Print Management
BASIC_GRAYSCALE_IMAGE_BOX	Print Management
BASIC_PRINT_IMAGE_OVERLAY_BOX	Print Management
BASIC_STRUCTURED_DISPLAY	Storage
BREAST_IMAGING_RPI_QUERY	Relevant Patient Information Query
BREAST_TOMO_IMAGE_STORAGE	Storage
CARDIAC_RPI_QUERY	Relevant Patient Information Query
CHEST_CAD_SR	Storage

Merge DICOM Service Parameter	DICOM Service Class
COLON_CAD_SR	Storage
COLOR_PALETTE_FIND	Color Palette Query/Retrieve
COLOR_PALETTE_GET	Color Palette Query/Retrieve
COLOR_PALETTE_MOVE	Color Palette Query/Retrieve
COLOR_PALETTE_STORAGE	Color Palette Storage
COMPOSITE_INST_RET_NO_BULK_GET	Query/Retrieve
COMPOSITE_INSTANCE_ROOT_RET_GET	Query/Retrieve
COMPOSITE_INSTANCE_ROOT_RET_MOVE	Query/Retrieve
DEFORMABLE_SPATIAL_REGISTRATION	Storage
DETACHED_INTERP_MANAGEMENT	Results Management
DETACHED_PATIENT_MANAGEMENT	Patient Management
DETACHED_RESULTS_MANAGEMENT	Results Management
DETACHED_STUDY_MANAGEMENT	Study Management
DETACHED_VISIT_MANAGEMENT	Patient Management
DICOMDIR	Media Storage
ENCAPSULATED_CDA	Storage
ENHANCED_CT_IMAGE	Storage
ENHANCED_MR_COLOR_IMAGE	Storage
ENHANCED_MR_IMAGE	Storage
ENHANCED_PET_IMAGE	Storage
ENHANCED_US_VOLUME	Storage
ENHANCED_XA_IMAGE	Storage
ENHANCED_XRF_IMAGE	Storage
G_P_PERFORMED_PROCEDURE_STEP	Study Management
G_P_SCHEDULED_PROCEDURE_STEP	Study Management
G_P_WORKLIST	Basic Worklist Management
GENERAL_AUDIO_WAVEFORM	Storage

Merge DICOM Service Parameter	DICOM Service Class
GENERAL_RPI_QUERY	Relevant Patient Information Query
GENERIC_IMPLANT_TEMPLATE	Storage
GENERIC_IMPLANT_TEMPLATE_FIND	Query/Retrieve
GENERIC_IMPLANT_TEMPLATE_GET	Query/Retrieve
GENERIC_IMPLANT_TEMPLATE_MOVE	Query/Retrieve
HANGING_PROTOCOL	Hanging Protocol Storage
HANGING_PROTOCOL_FIND	Hanging Protocol Query/Retrieve
HANGING_PROTOCOL_GET	Hanging Protocol Query/Retrieve
HANGING_PROTOCOL_MOVE	Hanging Protocol Query/Retrieve
IMAGE_OVERLAY_BOX_RETIRED	Print Management
IMPLANT_ASSEMBLY_TEMPLATE	Storage
IMPLANT_ASSEMBLY_TEMPLATE_FIND	Implant Template Query/Retrieve
IMPLANT_ASSEMBLY_TEMPLATE_GET	Implant Template Query/Retrieve
IMPLANT_ASSEMBLY_TEMPLATE_MOVE	Implant Template Query/Retrieve
IMPLANT_TEMPLATE_GROUP	Storage
IMPLANT_TEMPLATE_GROUP_FIND	Implant Template Query/Retrieve
IMPLANT_TEMPLATE_GROUP_GET	Implant Template Query/Retrieve
IMPLANT_TEMPLATE_GROUP_MOVE	Implant Template Query/Retrieve
IMPLANTATION_PLAN_SR_DOCUMENT	Storage
INSTANCE_AVAIL_NOTIFICATION	Instance Availability Notification
INTRAOCULAR_LENS_CALCULATIONS	Storage
KERATOMETRY_MEASUREMENTS	Storage
KEY_OBJECT_SELECTION_DOC	Storage

Merge DICOM Service Parameter	DICOM Service Class
LENSOMETRY_MEASUREMENTS	Storage
MACULAR_GRID_THIICKNESS_VOLUME	Storage
MAMMOGRAPHY_CAD_SR	Storage
MEDIA_CREATION_MANAGEMENT	Media Creation Management
MODALITY_WORKLIST_FIND	Basic Worklist Management
MR_SPECTROSCOPY	Storage
OPHT_VIS_FIELD_STATIC_PERIM_MEAS	Storage
OPHTHALMIC_AXIAL_MEASUREMENTS	Storage
OPHTHALMIC_TOMOGRAPHY_IMAGE	Storage
OPM_THICKNESS_MAP	Storage
PATIENT_ROOT_QR_FIND	Query/Retrieve
PATIENT_ROOT_QR_GET	Query/Retrieve
PATIENT_ROOT_QR_MOVE	Query/Retrieve
PATIENT_STUDY_ONLY_QR_FIND	Query/Retrieve
PATIENT_STUDY_ONLY_QR_GET	Query/Retrieve
PATIENT_STUDY_ONLY_QR_MOVE	Query/Retrieve
PERFORMED_PROCEDURE_STEP	Study Management
PERFORMED_PROCEDURE_STEP_NOTIFY	Study Management
PERFORMED_PROCEDURE_STEP_RETR	Study Management
PRESENTATION_LUT	Print Management
PRINT_JOB	Print Management
PRINT_QUEUE_MANAGEMENT	Print Management
PRINTER	Print Management
PRINTER_CONFIGURATION	Print Management
PROCEDURAL_EVENT_LOGGING	Application Event Logging
PROCEDURE_LOG	Storage
PRODUCT_CHARACTERISTICS_QUERY	Substance Administration Query

Merge DICOM Service Parameter	DICOM Service Class
PULL_PRINT_REQUEST	Print Management
RAW_DATA	Storage
REAL_WORLD_VALUE_MAPPING	Storage
REFERENCED_IMAGE_BOX	Print Management
RESPIRATORY_WAVEFORM	Storage
RT_BEAMS_DELIVERY_INSTRUCTION	Storage
RT_CONVENTIONAL_MACHINE_VERIF	RT Machine Verification
RT_ION_MACHINE_VERIF	RT Machine Verification
SC_MULTIFRAME_GRAYSCALE_BYTE	Storage
SC_MULTIFRAME_GRAYSCALE_WORD	Storage
SC_MULTIFRAME_SINGLE_BIT	Storage
SC_MULTIFRAME_TRUE_COLOR	Storage
SEGMENTATION	Storage
SPATIAL_FIDUCIALS	Storage
SPATIAL_REGISTRATION	Storage
SPECTACLE_PRESCRIPTION_REPORT	Storage
STANDARD_BASIC_TEXT_SR	Storage
STANDARD_BLENDING_SOFTCOPY_PS	Storage
STANDARD_COLOR_SOFTCOPY_PS	Storage
STANDARD_COMPREHENSIVE_SR	Storage
STANDARD_CR	Storage
STANDARD_CT	Storage
STANDARD_CURVE	Storage
STANDARD_DX_PRESENT	Storage
STANDARD_DX_PROCESS	Storage
STANDARD_ECHO	Verification
STANDARD_ENCAPSULATED_PDF	Storage
STANDARD_ENHANCED_SR	Storage

Merge DICOM Service Parameter	DICOM Service Class
STANDARD_GRAYSCALE_SOFTCOPY_PS	Storage
STANDARD_HARDCOPY_COLOR	Storage
STANDARD_HARDCOPY_GRAYSCALE	Storage
STANDARD_IO_PRESENT	Storage
STANDARD_IO_PROCESS	Storage
STANDARD_IVOCT_PRESENT	Storage
STANDARD_IVOCT_PROCESS	Storage
STANDARD_MG_PRESENT	Storage
STANDARD_MG_PROCESS	Storage
STANDARD_MODALITY_LUT	Storage
STANDARD_MR	Storage
STANDARD_NM	Storage
STANDARD_NM_RETIRED	Storage
STANDARD_OPHTHALMIC_16_BIT	Storage
STANDARD_OPHTHALMIC_8_BIT	Storage
STANDARD_OVERLAY	Storage
STANDARD_PET	Storage
STANDARD_PET_CURVE	Storage
STANDARD_PRINT_STORAGE	Storage
STANDARD_PSEUDOCOLOR_SOFTCOPY_PS	Storage
STANDARD_RT_BEAMS_TREAT	Storage
STANDARD_RT_BRACHY_TREAT	Storage
STANDARD_RT_DOSE	Storage
STANDARD_RT_IMAGE	Storage
STANDARD_RT_ION_BEAMS_TREAT	Storage
STANDARD_RT_ION_PLAN	Storage
STANDARD_RT_PLAN	Storage
STANDARD_RT_STRUCTURE_SET	Storage

Merge DICOM Service Parameter	DICOM Service Class
STANDARD_RT_TREAT_SUM	Storage
STANDARD_SEC_CAPTURE	Storage
STANDARD_US	Storage
STANDARD_US_MF	Storage
STANDARD_US_MF_RETIRED	Storage
STANDARD_US_RETIRED	Storage
STANDARD_VIDEO_ENDOSCOPIC	Storage
STANDARD_VIDEO_MICROSCOPIC	Storage
STANDARD_VIDEO_PHOTOGRAPHIC	Storage
STANDARD_VL_ENDOSCOPIC	Storage
STANDARD_VL_MICROSCOPIC	Storage
STANDARD_VL_PHOTOGRAPHIC	Storage
STANDARD_VL_SLIDE_MICROSCOPIC	Storage
STANDARD_VOI_LUT	Storage
STANDARD_WAVEFORM_12_LEAD_ECG	Storage
STANDARD_WAVEFORM_AMBULATORY_ECG	Storage
STANDARD_WAVEFORM_BASIC_VOICE_AU	Storage
STANDARD_WAVEFORM_CARDIAC_EP	Storage
STANDARD_WAVEFORM_GENERAL_ECG	Storage
STANDARD_WAVEFORM_HEMODYNAMIC	Storage
STANDARD_XRAY_ANGIO	Storage
STANDARD_XRAY_ANGIO_BIPLANE	Storage
STANDARD_XRAY_RF	Storage
STEREOMETRIC_RELATIONSHIP	Storage
STORAGE_COMMITMENT_PULL	Storage Commitment
STORAGE_COMMITMENT_PUSH	Storage Commitment
STUDY_COMPONENT_MANAGEMENT	Study Management
STUDY_CONTENT_NOTIFICATION	Study Content Notification

Merge DICOM Service Parameter	DICOM Service Class
STUDY_ROOT_QR_FIND	Query/Retrieve
STUDY_ROOT_QR_GET	Query/Retrieve
STUDY_ROOT_QR_MOVE	Query/Retrieve
SUBJ_REFRACTION_MEASUREMENTS	Storage
SUBSTANCE_ADMIN_LOGGING	Application Event Logging
SUBSTANCE_APPROVAL_QUERY	Substance Administration Query
SURFACE_SEGMENTATION	Storage
UPS_EVENT_SOP	Unified Procedure Step
UPS_PULL_SOP	Unified Procedure Step
UPS_PUSH_SOP	Unified Procedure Step
UPS_WATCH_SOP	Unified Procedure Step
VISUAL_ACUITY_MEASUREMENTS	Storage
VL_WHOLE_SLIDE_MICROSCOPY_IMAGE	Storage
VOI_LUT_BOX	Print Management
XA_XRF_GRAYSCALE_SOFTCOPY_PS	Storage
XRAY_3D_ANGIO_IMAGE	Storage
XRAY_3D_CRANIO_IMAGE	Storage
XRAY_RADATION_DOSE_SR	Storage
BASIC_COLOR_PRINT_MANAGEMENT (META_SOP)	Print Management
BASIC_GRAYSCALE_PRINT_MANAGEMENT (META_SOP)	Print Management
DETACHED_PATIENT_MANAGEMENT_META (META_SOP)	Patient Management
DETACHED_RESULTS_MANAGEMENT_META (META_SOP)	Results Management
G_P_WORKLIST_MANAGEMENT_META (META_SOP)	Basic Worklist Management
PULL_STORED_PRINT_MANAGEMENT (META_SOP)	Print Management

Merge DICOM Service Parameter	DICOM Service Class
REF_COLOR_PRINT_MANAGEMENT (META_SOP)	Print Management
REF_GRAYSCALE_PRINT_MANAGEMENT (META_SOP)	Print Management
STUDY_MANAGEMENT (META_SOP)	Study Management

Transfer syntax lists are contained in the service lists. The following is a list of the currently supported transfer syntaxes.

Table 32: Transfer Syntax List Parameters

Merge DICOM Transfer Syntax Parameter	Description
IMPLICIT_LITTLE_ENDIAN	Implicit VR Little Endian: Default Transfer Syntax for DICOM
IMPLICIT_BIG_ENDIAN	Implicit VR Big Endian
EXPLICIT_LITTLE_ENDIAN	Explicit VR Little Endian
EXPLICIT_BIG_ENDIAN	Explicit VR Big Endian
RLE	Run length Encoding
DEFLATED_EXPLICIT_LITTLE_ENDIAN	Deflated Explicit VR Little Endian
JPEG_BASELINE	JPEG Baseline (Process 1): Default Transfer Syntax for Lossy JPEG 8 Bit Image Compression
JPEG_EXTENDED_2_4	JPEG Extended (Process 2 & 4): Default Transfer Syntax for Lossy JPEG 12 Bit Image Compression (Process 4 only)
JPEG_EXTENDED_3_5	JPEG Extended (Process 3 & 5)
JPEG_SPEC_NON_HIER_6_8	JPEG Spectral Selection, Non-Hierarchical (Process 6 & 8)
JPEG_SPEC_NON_HIER_7_9	JPEG Spectral Selection, Non-Hierarchical (Process 7 & 9)
JPEG_FULL_PROG_NON_HIER_10_12	JPEG Full Progression, Non-Hierarchical (Process 10 & 12)
JPEG_FULL_PROG_NON_HIER_11_13	JPEG Full Progression, Non-Hierarchical (Process 11 & 13)
JPEG_LOSSLESS_NON_HIER_14	JPEG Lossless, Non-Hierarchical (Process 14)
JPEG_LOSSLESS_NON_HIER_15	JPEG Lossless, Non-Hierarchical (Process 15)

Merge DICOM Transfer Syntax Parameter	Description
JPEG_EXTENDED_HIER_16_18	JPEG Extended, Hierarchical (Process 16 & 18)
JPEG_EXTENDED_HIER_17_19	JPEG Extended, Hierarchical (Process 17 & 19)
JPEG_SPEC_HIER_20_22	JPEG Spectral Selection, Hierarchical (Process 20 & 22)
JPEG_SPEC_HIER_21_23	JPEG Spectral Selection, Hierarchical (Process 21 & 23)
JPEG_FULL_PROG_HIER_24_26	JPEG Full Progression, Hierarchical (Process 24 & 26)
JPEG_FULL_PROG_HIER_25_27	JPEG Full Progression, Hierarchical (Process 25 & 27)
JPEG_LOSSLESS_HIER_28	JPEG Lossless, Hierarchical (Process 28)
JPEG_LOSSLESS_HIER_29	JPEG Lossless, Hierarchical (Process 29)
JPEG_LOSSLESS_HIER_14	JPEG Lossless, Hierarchical, First-Order Prediction (Process 14 [Selection Value 1]): Default Transfer Syntax for Lossless JPEG Image Compression
JPEG_2000_LOSSLESS_ONLY	JPEG 2000, Lossless
JPEG_2000	JPEG 2000, Lossless or Lossy
JPEG_LS_LOSSLESS	JPEG LS Lossless
JPEG_LS_LOSSY	JPEG LS Lossy (Near-Lossless)
JPEG_2000_MC_LOSSLESS_ONLY	JPEG 2000 Part 2 Multi-component Image Compression (Lossless Only)
JPEG_2000_MC	JPEG 2000 Part 2 Multi-component Image Compression
JPIP_REFERENCED	JPIP Referenced
JPIP_REFERENCED_DEFLATE	JPIP Referenced Deflate
MPEG2_MPHL	MPEG2 Main Profile @ High Level
MPEG2_MPML	MPEG2 Main Profile @ Main Level
MPEG4_AVC_H264_HP_LEVEL_4_1	MPEG-4 AVC/H.264 High Profile / Level 4.1
MPEG4_AVC_H264_BDC_HP_LEVEL_4_1	MPEG-4 AVC/H.264 BDcompatible High Profile / Level 4.1

Merge DICOM Transfer Syntax Parameter	Description
PRIVATE_SYNTAX_1	Private transfer syntax 1 with the characteristics specified by the PRIVATE_SYNTAX_1_LITTLE_ENDIAN, PRIVATE_SYNTAX_1_EXPLICIT_VR, and PRIVATE_SYNTAX_1_ENCAPSULATED configuration options.
PRIVATE_SYNTAX_2	Private transfer syntax 2 with the characteristics specified by the PRIVATE_SYNTAX_2_LITTLE_ENDIAN, PRIVATE_SYNTAX_2_EXPLICIT_VR, and PRIVATE_SYNTAX_2_ENCAPSULATED configuration options.

System Profile

The System Profile is used to define system-wide parameters. These parameters apply across all associations with other DICOM application entities. The location of this file is provided by the `MERGECOM_3_PROFILE` parameter of the `[MergeCOM3]` section of the `MERGE.INI` file.

The following are a few notes to keep in mind concerning the System Profile:

- You must specify your own unique DICOM Implementation Class UID and place it in this file along with an optional Implementation Version. These need to be documented in your DICOM conformance statement.
- There are several exception options specified at both the association and DIMSE levels of DICOM communication. You should not have to modify these options in normal circumstances and doing so could make your application non DICOM conformant.
- The DICOM Upper Layer section network time-outs can be modified. This is useful on slower or less-predictable networks (e.g., WAN's).
- The section of the System Profile dealing with transport parameters is important. This is where you specify the **TCP/IP listen port** for a DICOM server (SCP) application, along with the **number of simultaneous associations** your server will support over this port.

Table 33 defines how each parameter should be defined within the system profile.

Table 33: System profile parameters

Name	Section	Description
ACCEPT_ANY_APPLICATION_TITLE †	ASSOC_PARMS	If set to YES, the remote system need not specify a correct DICOM application title when requesting an association. If set to NO a correct application title must be used. When this value is set to YES, the toolkit will report the remote application as connecting to the first application registered. DEFAULT: NO
ACCEPT_ANY_CONTEXT_NAME †	ASSOC_PARMS	If set to YES, the remote system need not specify the LOCAL_APPL_CONTEXT_NAME when requesting an association. If set to NO, the correct context name must be used. DEFAULT: NO
ACCEPT_ANY_HOSTNAME	ASSOC_PARMS	If set to YES, the toolkit will not check if applications connecting to an SCP can have their hostname resolved through the SCP's host file or domain name server. If set to NO, the toolkit will automatically reject associations from unknown hosts. DEFAULT: NO
ACCEPT_ANY_PRESENTATION_CONTEXT †	ASSOC_PARMS	If set to YES, the tool kit will not validate that the presentation context ID contained in a message's PDU header information matches the ID of the presentation context negotiated for the type of message contained in the PDU. If set to NO, the toolkit will abort associations when these values do not match. DEFAULT: NO
ACCEPT_DIFFERENT_IC_UID †	ASSOC_PARMS	If set to NO, the remote system must specify the local IMPLEMENTATION_CLASS_UID when requesting an association. If set to YES, a different implementation class UID may be used. DEFAULT: YES

Name	Section	Description
ACCEPT_DIFFERENT_VERSION †	ASSOC_PARMS	If set to NO, the remote system must specify the local IMPLEMENTATION_VERSION when requesting an association. If set to YES, a different implementation version may be used. DEFAULT: YES
ACCEPT_MULTIPLE_PRES_CONTEXTS	ASSOC_PARMS	If set to YES, SCP applications will allow multiple presentation contexts to be negotiation for a single DICOM service. If set to NO, an SCP will only accept a single presentation context for a DICOM service. DEFAULT:YES
EXPLICIT_BIG_ENDIAN_SYNTAX	ASSOC_PARMS	This value defines the UID of the explicit VR big endian transfer syntax. DEFAULT: 1.2.840.10008.1.2.2
EXPLICIT_LITTLE_ENDIAN_SYNTAX	ASSOC_PARMS	This value defines the UID of the explicit VR little endian transfer syntax. DEFAULT: 1.2.840.10008.1.2.1
HARD_CLOSE_TCP_IP_CONNECTION	ASSOC_PARMS	This parameter specifies how TCP/IP connections are closed by the toolkit. When set to YES, TCP/IP connections are instantaneously closed with a RST packet. When set to NO, TCP/IP connections are closed gracefully with a FIN packet. Note, that in the NO case the toolkit must wait for an operating system dependent amount of time for the response to the FIN packet. DEFAULT: YES
IMPLEMENTATION_CLASS_UID	ASSOC_PARMS	The DICOM Implementation Class UID (as specified in your DICOM conformance statement).
IMPLEMENTATION_VERSION	ASSOC_PARMS	The Implementation Version Number (as specified in your DICOM conformance statement).
DEFLATED_EXPLICIT_LITTLE_ENDIAN _SYNTAX	ASSOC_PARMS	This value defined the UID of the Deflated explicit VR little endian transfer syntax. DEFAULT: 1.2.840.10008.1.2.1.99

Name	Section	Description
IMPLICIT_BIG_ENDIAN_SYNTAX	ASSOC_PARMS	The implicit big endian transfer syntax is not defined by the DICOM standard. This value is provided to supply compatibility with private implementations. DEFAULT: <none>
IMPLICIT_LITTLE_ENDIAN_SYNTAX	ASSOC_PARMS	The implicit little endian transfer syntax is the default network transfer syntax of the DICOM standard. The implicit little endian transfer syntax must always be defined. DEFAULT: 1.2.840.10008.1.2
LICENSE	ASSOC_PARMS	The Merge DICOM Toolkit license number that was supplied when the toolkit was purchased.
RLE_SYNTAX	ASSOC_PARMS	This value defines the UID of the RLE transfer syntax. DEFAULT: 1.2.840.10008.1.2.5
JPEG_BASELINE_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Baseline (Process 1) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.50
JPEG_EXTENDED_2_4_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Extended (Process 2 & 4) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.51
JPEG_EXTENDED_3_5_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Extended (Process 3 & 5) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.52
JPEG_SPEC_NON_HIER_6_8_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Spectral Selection, Non-Hierarchical (Process 6 & 8) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.53
JPEG_SPEC_NON_HIER_7_9_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Spectral Selection, Non-Hierarchical (Process 7 & 9) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.54

Name	Section	Description
JPEG_FULL_PROG_NON_HIER_10_12 _SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Full Progression, Non-Hierarchical (Process 10 & 12) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.55
JPEG_FULL_PROG_NON_HIER_11_13 _SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Full Progression, Non-Hierarchical (Process 11 & 13) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.56
JPEG_LOSSLESS_NON_HIER_14_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Lossless, Non-Hierarchical (Process 14) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.57
JPEG_LOSSLESS_NON_HIER_15_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Lossless, Non-Hierarchical (Process 15) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.58
PEG_EXTENDED_HIER_16_18_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Extended, Hierarchical (Process 16 & 18) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.59
JPEG_EXTENDED_HIER_17_19_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Extended, Hierarchical (Process 17 & 19) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.60
JPEG_SPEC_HIER_20_22_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Spectral Selection, Hierarchical (Process 20 & 22) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.61
JPEG_SPEC_HIER_21_23_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Spectral Selection, Hierarchical (Process 21 & 23)transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.62
JPEG_FULL_PROG_HIER_24_26_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Full Progression, Hierarchical (Process 24 & 26) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.63

Name	Section	Description
JPEG_FULL_PROG_HIER_25_27_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Full Progression, Hierarchical (Process 25 & 27) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.64
JPEG_LOSSLESS_HIER_28_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Lossless, Hierarchical (Process 28) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.65
JPEG_LOSSLESS_HIER_29_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Lossless, Hierarchical (Process 29) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.66
JPEG_LOSSLESS_HIER_14_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG Lossless, Hierarchical, First-Order Prediction (Process 14, Selection Value 1) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.70
JPEG_2000_LOSSLESS_ONLY_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG 2000, Lossless. DEFAULT: 1.2.840.10008.1.2.4.90
JPEG_2000_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG 2000. DEFAULT: 1.2.840.10008.1.2.4.91
JPEG_2000_MC_LOSSLESS_ONLY_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG 2000 Part 2 Multi-component Image Compression (Lossless Only) DEFAULT: "1.2.840.10008.1.2.4.93"
JPEG_2000_MC_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG 2000 Part 2 Multi-component Image Compression DEFAULT: "1.2.840.10008.1.2.4.92"
JPEG_LS_LOSSLESS_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG LS Lossless. DEFAULT: 1.2.840.10008.1.2.4.80
JPEG_LS_LOSSY_SYNTAX	ASSOC_PARMS	This value defines the UID for JPEG LS Lossy (Near Lossless). DEFAULT: 1.2.840.10008.1.2.4.81

Name	Section	Description
MPEG2_MPML_SYNTAX	ASSOC_PARMS	This value defines the UID for MPEG2 Main Profile @ Main Level. DEFAULT: 1.2.840.10008.1.2.4.100
LOCAL_APPL_CONTEXT_NAME	ASSOC_PARMS	The DICOM Application Context Name (UID) (as specified in the DICOM Standard). DEFAULT: "1.2.840.10008.3.1.1.1"
PDU_MAXIMUM_LENGTH *	ASSOC_PARMS	The maximum size of DICOM Protocol Data Units that can be received by this Merge DICOM implementation. This value is used during association negotiation to specify the maximum sized PDUs that your application can receive. DEFAULT: 28K MINIMUM: 4K MAXIMUM: NONE
PRIVATE_SYNTAX_1_SYNTAX	ASSOC_PARMS	The unique identifier (UID) Merge DICOM will use to identify private transfer syntax 1. When this value is set to "<none>", private transfer syntax support is shut off. DEFAULT: <none>
PRIVATE_SYNTAX_1_LITTLE_ENDIAN	ASSOC_PARMS	When set to YES, Merge DICOM will interpret private transfer syntax 1 as being encoded in little endian format. DEFAULT: YES
PRIVATE_SYNTAX_1_EXPLICIT_VR	ASSOC_PARMS	When set to YES, Merge DICOM will interpret private transfer syntax 1 as being encoded in explicit VR format. DEFAULT: YES
PRIVATE_SYNTAX_1_ENCAPSULATED	ASSOC_PARMS	When set to YES, Merge DICOM will interpret private transfer syntax 1 as having its pixel data tag (7fe0,0010) being encoded as undefined length in the same manner as the JPEG and RLE transfer syntaxes are encoded. DEFAULT: NO

Name	Section	Description
PRIVATE_SYNTAX_2_SYNTAX	ASSOC_PARMS	The unique identifier (UID) Merge DICOM will use to identify private transfer syntax 2. When this value is set to "<none>", private transfer syntax support is shut off. DEFAULT: <none>
PRIVATE_SYNTAX_2_LITTLE_ENDIAN	ASSOC_PARMS	When set to YES, Merge DICOM will interpret private transfer syntax 2 as being encoded in little endian format. DEFAULT: YES
PRIVATE_SYNTAX_2_EXPLICIT_VR	ASSOC_PARMS	When set to YES, Merge DICOM will interpret private transfer syntax 2 as being encoded in explicit VR format. DEFAULT: YES
PRIVATE_SYNTAX_2_ENCAPSULATED	ASSOC_PARMS	When set to YES, Merge DICOM will interpret private transfer syntax 2 as having its pixel data tag (7fe0,0010) being encoded as undefined length in the same manner as the JPEG and RLE transfer syntaxes are encoded. DEFAULT: NO
INITIATOR_NAME †	DIMSE_PARMS	The DICOM standard has retired the old ACR/NEMA Initiator Name attribute in command messages. To generated such an attribute in command messages, specify an initiator name. <none> means do not put initiator name in messages. DEFAULT: <none>
RECEIVER_NAME †	DIMSE_PARMS	The DICOM standard has retired the old ACR/NEMA Receiver Name attribute in command messages. To generated such an attribute in command messages, specify a receiver name. <none> means do not put receiver name in messages. DEFAULT: <none>

Name	Section	Description
SEND_ECHO_PRIORITY †	DIMSE_PARMS	The DICOM standard has retired the message priority attribute in echo command messages. To generated such an attribute in command messages, specify YES. To NOT use message priority in echo messages, specify NO. DEFAULT: NO
SEND_LENGTH_TO_END †	DIMSE_PARMS	The DICOM standard has retired the old Group-Length-To-End attribute in command messages. To generate such an attribute in command messages, specify YES. If you do not want to generate Group-Length-To-End, specify NO. DEFAULT: NO
SEND_MSG_ID_RESPONSE †	DIMSE_PARMS	The DICOM standard has retired the message ID attribute in response command messages. To generated such an attribute in command messages, specify YES. To NOT use message ID in response messages, specify NO. DEFAULT: NO
SEND_RECOGNITION_CODE †	DIMSE_PARMS	The DICOM standard has retired the old Recognition Code attribute in command messages. To generated such an attribute in command messages, specify YES. If you do not want to generate such an attribute, specify NO, ACR-NEMA 2.0. DEFAULT: NO
SEND_RESPONSE_PRIORITY †	DIMSE_PARMS	The DICOM standard has retired the message priority attribute in response messages. To generated such an attribute in response messages, specify YES. To NOT use message priority in response messages, specify NO. DEFAULT: NO

Name	Section	Description
SEND_SOP_CLASS_UID †	DIMSE_PARMS	Certain DICOM service classes demand that the affected SOP class UID be present in the message. To prevent the library from insuring that this is done, specify NO. To insure that Affected SOP class UID is present, specify YES. DEFAULT: YES
SEND_SOP_INSTANCE_UID †	DIMSE_PARMS	Certain DICOM service classes demand that the affected SOP instance UID be present in the message. To prevent the library from insuring that this is done, specify NO. To insure that Affected SOP instance UID is present, specify YES. DEFAULT: YES
ARTIM_TIMEOUT	DUL_PARMS	The number of seconds to use as a time out waiting for an association request or waiting for the peer to shut down an association. DEFAULT: 30
ASSOC_REPLY_TIMEOUT	DUL_PARMS	The number of seconds to wait for a reply to an associate request. DEFAULT: 15.
CONNECT_TIMEOUT	DUL_PARMS	The number of seconds to wait for a network connect to be accepted. DEFAULT: 15.
INACTIVITY_TIMEOUT	DUL_PARMS	The number of seconds to wait in between packets of data received over the network after the initial packet of data in a message is received. Used by the read method. DEFAULT: 15.
INSURE_EVEN_UID_LENGTH †	DUL_PARMS	Set to NO, if odd-length UIDs in PDU's should NOT be padded with a NULL to insure even length unique Ids. Set to YES to insure even UIDs in PDUs. DEFAULT: NO
RELEASE_TIMEOUT	DUL_PARMS	The number of seconds to wait for a reply to an associate release. DEFAULT: 15.

Name	Section	Description
WRITE_TIMEOUT	DUL_PARMS	The number of seconds to wait for a network write to be accepted. DEFAULT: 15.
DICOMDIR_STREAM_STORAGE	MEDIA_PARMS	When set to yes, DICOMDIRs read in leave their directory records internally in "stream" format, and are not parsed until the directory record is referenced. This can greatly reduce memory usage when reading in large DICOMDIRs when the entire DICOMDIR is not referenced. Default: NO
EXPORT_PRIVATE_ATTRIBUTES_TO_MEDIA	MEDIA_PARMS	When set to NO, disable the exporting of private attributes in files written with the writeFile and writeFileByCallback methods. DEFAULT: YES
EXPORT_UN_VR_TO_MEDIA	MEDIA_PARMS	When set to NO, disable the exporting of attributes with a VR of UN in files written with the writeFile and writeFileByCallback methods. DEFAULT: YES
EXPORT_GROUP_LENGTHS_TO_MEDIA	MEDIA_PARMS	When set to NO, do not write group length attributes with MC_Write_File() and MC_Write_File_By_Callback() DEFAULT: NO
EXPORT_UNDEFINED_LENGTH_SQ_IN_DICOMDIR *	MEDIA_PARMS	When set to NO, DICOMDIRs written with MC_Write_File() are created with their sequence attributes having defined lengths. Setting this option to Yes will increase performance. DEFAULT: YES
ALLOW_COMMA_IN_DS_FL_FD_STRINGS	MESSAGE_PARMS	When set to Yes, a comma or a period will be allowed in String values passed to appendValue, putValue or setValue for attributes with a VR of DS, FL or FD. When set to No, only a period will be acceptable as a decimal separator. Note that the toolkit will always ensure that DS attributes use a period decimal separator when streaming to the network or to a file, regardless of current locale settings. DEFAULT: NO

Name	Section	Description
ALLOW_INVALID_PRIVATE_ATTRIBUTES	MESSAGE_PARMS	When reading messages or file objects, this parameter specifies if private attributes encoded in an invalid format should be ignored or parsed. DEFAULT: NO
ALLOW_INVALID_PRIVATE_CREATOR _CODES	MESSAGE_PARMS	When reading messages or file objects, this parameter specifies if private creator codes encoded with invalid characters should be ignored or parsed. DEFAULT: NO
CALLBACK_MIN_DATA_SIZE	MESSAGE_PARMS	When using an MCattributeContainer to store large data such as pixel data, this option specifies the minimum size of value for which the callback should be used. This option was specifically added so pixel data contained in icons are not managed with a callback. DEFAULT: 1
COMPRESSION_LUM_FACTOR	MESSAGE_PARMS	Configuration Parameter for MC_Standard_Compressor. Value 0 through 255. 0 is the highest quality, giving a quantization table of all 1's. 32 corresponds to the standard quantization tables. For values between 0 and 128, the standard tables are scaled linearly. For values between 128 and 255, the standard tables are scaled non-linearly and the compression increases (and the quality decreases) by a very large amount. DEFAULT: 32
COMPRESSION_CHROM_FACTOR	MESSAGE_PARMS	Configuration Parameter for MC_Standard_Compressor.Value 0 through 255.The chrominance compression factor is used to adjust the default chrominance quantization table values. When ChromFactor is 32, the default chrominance quantization table values are used as is. A value of 255 corresponds to high compression, low quality. DEFAULT: 32

Name	Section	Description
COMPRESSION_ALLOW_FRAGS	MESSAGE_PARMS	Configuration Parameter for MC_Standard_Compressor. The Pegasus libraries allow compressed image data to be returned as it continues to compress more image data. This may result in an image frame having one or more fragments. This is perfectly legal, however some viewers may not be able to display the image if they do not support multiple fragments per frame.
COMPRESSION_WHEN_J2K_USE_LOSSY	MESSAGE_PARMS	Configuration Parameter for MC_Standard_Compressor. When JPEG_2000 is used as a transfer syntax, this could mean either lossy or lossless compression. This parameter specifies the intended syntax.
COMPRESSION_J2K_LOSSY_RATIO	MESSAGE_PARMS	Configuration Parameter for MC_Standard_Compressor. When JPEG_2000 with COMPRESSION_WHEN_J2K_USE_LOSSY = Yes, and COMPRESSION_J2K_LOSSY_USE_QUALITY = No, a ratio can be specified. The compressor attempts to reduce the image size to 1/COMPRESSION_J2K_LOSSY_RATIO.
COMPRESSION_J2K_LOSSY_QUALITY	MESSAGE_PARMS	Configuration Parameter for MC_Standard_Compressor. When JPEG_2000 with COMPRESSION_WHEN_J2K_USE_LOSSY = Yes, and COMPRESSION_J2K_LOSSY_USE_QUALITY = Yes, a quality can be specified. Valid values are 1 to 10, 1 being highest quality image.
COMPRESSION_J2K_LOSSY_USE_QUALITY	MESSAGE_PARMS	Configuration Parameter for MC_Standard_Compressor. When JPEG_2000 with COMPRESSION_WHEN_J2K_USE_LOSSY = Yes, this indicates which metric should be used for lossy compression, ratio or quality.

Name	Section	Description
COMPRESSION_RGB_TRANSFORM _FORMAT	MESSAGE_PARMS	This parameter allows the user to select the output format when doing Lossy JPEG compression of RGB images. The value can be set to YBR_FULL or YBR_FULL_422 to specify what photometric interpretation Merge DICOM should compress into when compressing RGB images. DEFAULT: YBR_FULL_422
CREATE_OFFSET_TABLE	MESSAGE_PARMS	This parameter specifies if an offset table is created when MC_Duplicate_Message is used to compress a DICOM message or file. It also specifies if an offset table is created when the MC_Set_Encapsulated_Value_From_ Function and MC_Set_Next_Encapsulated_Value_F rom_Function routines are used. DEFAULT: Yes
DEFLATE_COMPRESSION_LEVEL	MESSAGE_PARMS	Allows the compression level of deflate to be specified when using deflated explicit VR little endian transfer syntax. 0 is no compression, 1 is fastest, and 9 compresses best. DEFAULT: -1
DEFLATE_ALLOW_FLUSH	MESSAGE_PARMS	Allows deflate to flush data occasionally to limit buffering. DEFAULT: Yes
DICTIONARY_ACCESS	MESSAGE_PARMS	This parameter specifies whether or not the DICOM dictionary is to be loaded into memory, or accessed from the dictionary file. FILE means access information directly from the dictionary file. MEM means load the dictionary into memory and access it there. NOTE: This parameter is ignored if the dictionary is pre-compiled and loaded as part of the program (using the gendict utility) NOTE: Embedded systems must always use pre-compiled dictionary entries. DEFAULT: MEM

Name	Section	Description
DICTIONARY_FILE	MESSAGE_PARMS	This parameter specifies the name (path) of the DICOM dictionary. An absolute or relative path may be specified. NOTE: This parameter is ignored if the dictionary has been pre-compiled. DEFAULT: ../mc3msg/mrgcom3.dct
DUPLICATE_ENCAPSULATED_ICON	MESSAGE_PARMS	When duplicating to an encapsulated transfer syntax, this configuration value specifies whether an ICON IMAGE SEQUENCE should also be encapsulated.
ELIMINATE_ITEM_REFERENCES *	MESSAGE_PARMS	This parameter specifies the behavior of the removeMessageValues::MCdimseMessage, cleanup::MCdataSet, cleanup::MCfile, cleanup::MCitem, removeFileValues::MCfile, and removeAllValues::McatrtibuteSet methods. If this parameter is set to YES, the above methods will search for references in every currently open attribute set to delete when they encounter an item to free within an attribute set. DEFAULT: YES.
EMPTY_PRIVATE_CREATOR_CODES	MESSAGE_PARMS	If set to NO, private creator codes contained in attribute sets are not emptied when the removeFileValues or removeMessageValues method calls are made. DEFAULT: YES
EXPORT_PRIVATE_ATTRIBUTES_TO_NETWORK	MESSAGE_PARMS	When set to NO, disable the exporting of private attributes in messages written to the network with the sendRequestMessage or sendResponseMessage methods. DEFAULT: YES
EXPORT_UN_VR_TO_NETWORK	MESSAGE_PARMS	When set to NO, disable the exporting of attributes with a VR of UN in messages written to the network with the sendRequestMessage or sendResponseMessage methods. DEFAULT: YES

Name	Section	Description
EXPORT_UNDEFINED_LENGTH_SQ *	MESSAGE_PARMS	If YES, messages transferred over the network or written to disk have their sequence attributes encoded as undefined length. This increases performance of the library. DEFAULT: NO
EXPORT_GROUP_LENGTHS_TO_NETWORK *	MESSAGE_PARMS	When set to NO, do not export group length attributes when using the MC_Send_Request_Message() and MC_Send_Response_Message() functions DEFAULT: NO
FLATE_GROW_OUTPUT_BUF_SIZE *	MESSAGE_PARMS	The size that the output buffer of deflate or inflate should grow when it's size is insufficient. An Info message is logged each time the buffer grows. DEFAULT: 1024
FORCE_OPEN_EMPTY_ITEM *	MESSAGE_PARMS	When set to YES, the MCitem constructor that provides an item name will act similar to the MCitem constructer that is used to create an "empty" item. The up-front performance cost of the former constructer will be reduced, but the amount of validation done when adding tags to the item is reduced. Setting this value to YES will also improve the performance of the DICOMDIR directory methods. This configuration value does not have any effect on embedded platforms. DEFAULT: NO

Name	Section	Description
IGNORE_JPEG_BAD_SUFFIX	MESSAGE_PARMS	Config Parameter for MC_Standard_Decompressor to deal with lossless JPEG images whose suffix have been invalidly written according to the JPEG specification. These images have a 16-zero-bit suffix following a -32768 prefix where the JPEG spec says the suffix is omitted following a -32768 prefix. The following are the valid settings: -1 = Default, fail on these images 0 = Ignore when user detects such images 1 = Let the toolkit detect and ignore automatically
LARGE_DATA_SIZE	MESSAGE_PARMS	Defines "Large Data" to the toolkit. "Large Data" is defined as an attribute value which has a length of LARGE_DATA_SIZE or more. DEFAULT: 200.
LARGE_DATA_STORE	MESSAGE_PARMS	This parameter specifies where "Large Data" values should be stored. FILE means store the values in temporary files. MEM means store the values in memory. NOTE: Embedded systems should ignore this parameter and always use MEM. DEFAULT: MEM
MSG_INFO_FILE	MESSAGE_PARMS	This parameter specifies the name (path) of the DICOM message information file. An absolute or relative path may be specified. DEFAULT: ../mc3msg/mrgcom3.msg
NULL_TYPE3_VALIDATION	MESSAGE_PARMS	This parameter specifies how the toolkit will validate a single NULL value in a type 3 attribute with VM>1. DEFAULT: ERR
OBOW_BUFFER_SIZE	MESSAGE_PARMS	This parameter specifies the number of bytes of "Large Data" that should be buffered before it is written to disk. This value is only used when the parameter LARGE_DATA_STORE is set to FILE. DEFAULT: 4096

Name	Section	Description
PEGASUS_OP_*_NAME	MESSAGE_PARMS	When using your own Pegasus license to remove the 3 frames/second limit, this should have the company name that was used to generate your Pegasus license.
PEGASUS_OP_*_REGISTRATION	MESSAGE_PARMS	When using your own Pegasus license to remove the 3 frames/second limit, this should have the registration code that goes with it's respective PEGASUS_OP_*_NAME.
PEGASUS_OPCODE_PATH	MESSAGE_PARMS	This parameter specifies the directory where Pegasus opcode DLLs are to be loaded from. The opcode DLL refers to files like picn6220 and not the dispatcher DLL picn20. If the option is empty, the SSM/DLL is loaded from the same directory as the dispatcher DLL. If these files are not found, opcode SSM/DLL is loaded using the directory order Windows uses when loading DLLs. The SSM/DLL is loaded from the current directory if '.' is specified. DEFAULT: (empty)
REMOVE_PADDING_CHARS	MESSAGE_PARMS	When set to Yes, Merge DICOM will remove space padding characters from all text based attributes. This removal will occur when the attribute is encoded with one of the MCattributeSet append, put or set methods, or when the attribute is read with one of the streaming or network read methods. DEFAULT: No
RETURN_COMMA_IN_DS_FL_FD_STRINGS	MESSAGE_PARMS	When set to Yes, Merge DICOM will return a comma character as the decimal separator in a value when a String value is requested in the getFirstValue or getNextValue methods for an attribute with a VR of DS, FL, or FD. When set to No, a period will always be returned for the decimal separator. Note that DS values will always be properly encoded with a period in DICOM message objects. DEFAULT: No

Name	Section	Description
TEMP_FILE_DIRECTORY	MESSAGE_PARMS	This parameter specifies the directory in which temporary files should be created. This parameter is used only if LARGE_DATA_STORE = FILE. An absolute or relative path may be specified. DEFAULT: ./
UN_VR_CODE	MESSAGE_PARMS	VR Code to use for attributes with unknown VRs. This may be set to 'OB' if an implementation does not understand 'UN'. DEFAULT: UN VALID VALUES: UN, OB
WORK_BUFFER_SIZE *	MESSAGE_PARMS	This parameter specifies the amount of data that is buffered in the toolkit before being stored internally or passed to a user's callback method. This option impacts the streamIn, streamOut, sendRequestMessage, sendResponseMessage, read, readFile, readFileBypassLargeData, readFileUpToTag, writeFile and writeFileByCallback methods. Setting this option to values larger than 28K will in most cases cause the toolkit to use the operating system's memory management scheme instead of the toolkit's internal mechanism. DEFAULT: 28K
CAPTURE_FILE	TRANSPORT_PARMS	This parameter specifies the base name to use for capture files. (Capture files are generated if the NETWORK_CAPTURE value is set to Yes.) If only one capture file is requested (see NUMBER_OF_CAP_FILES), the capture file will have the name specified. If more than one is requested, nnn will be appended to the base file name specified (e.g., merge001.cap) DEFAULT: merge.cap (in the current directory)

Name	Section	Description
CAPTURE_FILE_SIZE	TRANSPORT_PARMS	This parameter specifies the maximum size (in kilobytes) that capture files are allowed to grow. (Capture files are generated if the NETWORK_CAPTURE value is set to Yes.) If more than one capture file is requested (see NUMBER_OF_CAP_FILES), each file generated will have this maximum size. If a value less than 1 is specified only one capture file of unlimited length will be generated. DEFAULT: 0
MAX_PENDING_CONNECTIONS	TRANSPORT_PARMS	This parameter specifies the maximum number of open listen channels. It's value is used as the second argument of a TCP listen() call. DEFAULT: 5
NETWORK_CAPTURE	TRANSPORT_PARMS	This parameter specifies whether or not network data should be captured in files suitable to be read by the MergeDPM utility. Use these parameters to customize the network capture: CAPTURE_FILE CAPTURE_FILE_SIZE NUMBER_OF_CAP_FILES REWRITE_CAPTURE_FILES DEFAULT: No
NUMBER_OF_CAP_FILES	TRANSPORT_PARMS	This parameter specifies the number of capture files to generate. (Capture files are generated if the NETWORK_CAPTURE value is set to Yes.) Each capture file generated will have maximum size specified by CAPTURE_FILE_SIZE. If multiple capture files is requested, the network data of each association will be captured in its own file. If CAPTURE_FILE_SIZE is less than 1 (unlimited size) this parameter's value is ignored. DEFAULT: 1

Name	Section	Description
REWRITE_CAPTURE_FILES	TRANSPORT_PARMS	This parameter specifies the whether or not capture files should be rewritten when all files have reached the maximum size specified by CAPTURE_FILE_SIZE. (Capture files are generated if the NETWORK_CAPTURE value is set to Yes.) If Yes is specified, the oldest written file will be rewritten. If No is specified and all requested files have been written (see NUMBER_OF_CAP_FILES), no more data will be captured. DEFAULT: Yes
MAX_PENDING_CONNECTIONS	TRANSPORT_PARMS	This parameter specifies the maximum number of open listen channels. It's value is used as the second argument of a TCP listen() call. DEFAULT: 5
TCPIP_DISABLE_NAGLE	TRANSPORT_PARMS	This parameter specifies if the Nagle Algorithm should be used when sending packets at the TCP/IP level. Most operating systems enable this by default. It allows small segments of data to delay sending a fixed amount of time to possibly be combined with other small segments and be sent as one larger packet. Disabling this may cause high network traffic.
TCPIP_LISTEN_PORT	TRANSPORT_PARMS	This parameter specifies the TCP/IP port on which server applications are to listen for associate requests. DEFAULT: 104
TCPIP_RECEIVE_BUFFER_SIZE *	TRANSPORT_PARMS	This parameter specifies the TCP/IP receive buffer size for each connection. For optimum performance, this value should be set slightly higher than the value for PDU_MAXIMUM_LENGTH. DEFAULT: 29K MAXIMUM: Operating System dependent

Name	Section	Description
TCPIP_SEND_BUFFER_SIZE *	TRANSPORT_PARMS	This parameter specifies the TCP/IP send buffer size for each connection. For optimum performance, this value should be set slightly higher than the value for PDU_MAXIMUM_LENGTH. DEFAULT: 29K MAXIMUM: Operating System dependent
IP_TYPE	TRANSPORT_PARMS	Sets the prefered IP type for network communications. When set to IPV4, only IPV4 connections are allowed. When set to IPV6, only IPV6 connects are allowed. When set to AVAILABLE, Merge DICOM will use IPV4 or IPV6 if it is enabled in the operating system. DEFAULT: IPv4

†**These options allow for non-standard DICOM operations**. Such exceptions, if used, should be noted in your DICOM conformance statement.
*Performance Tuning

Service Profile

The Service Profile is generated by Merge OEM and contains DICOM standard services and commands and is a useful reference (along with the `message.txt` file mentioned previously) to find the Merge DICOM names for the standard DICOM services and items. It is used by the library to negotiate the proper SOP Class UIDs and to access the binary dictionary and message information files when creating instances of message objects and validating messages.

In most cases, it will not be necessary to modify the Service Profile. However, if you are using an extended toolkit to create your own private services, you will need to add specifications for these private services to the Service Profile. See the *Merge DICOM Toolkit: Extended Toolkit Manual* for further details.

The location of the Service Profile is provided by the `MERGECOM_3_SERVICES` parameter of the `[MergeCOM3]` section of the `MERGE.INI` file.

Remember, the Service Profile is GENERATED by the Merge DICOM Profile Database Utilities at Merge OEM. Unless you are absolutely confident about changes being made, DO NOT CHANGE THE CONTENTS OF THIS FILE.

The Service Profile contains the following sections.

Table 34: Service profile parameters

Name	Description
[SERVICE_TABLE]	List of service names and numbers. This list registers every service available to an Application Entity. The parameters associated with [SERVICE_LIST] are NUMBER_OF_SERVICES_SUPPORTED (the number of service names that will be listed immediately following NUMBER_OF_SERVICES_SUPPORTED) and one entry for each supported service.
[<service_number>]	One section number for each of the above services registered in [SERVICE_TABLE]. Each section contains a Service Name, a DICOM SOP Class UID for the Service, a flag that tells whether it is a BASE or META Service (SOP) and a list of commands supported for that service.
[ITEM_TABLE]	One item name and number for each DICOM item that can be encoded in an attribute of Value representation SQ (Sequence of Items).

Appendix E: XML structure

The Merge DICOM Toolkit provides an API to convert a DICOM message, file or attribute set into an XML string. The following displays the basic structure of the XML string.

Base64 encoding of bulks and attributes with VR UN:

```
<?xml version="1.0" encoding="utf-8"?>
<DcmFile>
  <FileMetaInfo Service="STANDARD_SEC_CAPTURE"
       Command="C_STORE_RQ">
    <Attribute Tag="00020001" VR="OB" Name="File Meta
       Information Version" Length="2">AAE=</Attribute>
    <Attribute Tag="00020002" VR="UI" Name="Media Storage
       SOP Class UID" Length="25">...</Attribute>
    <Attribute Tag="00020003" VR="UI" Name="Media Storage
       SOP Instance UID" Length="29">...</Attribute>
    <Attribute Tag="00020010" VR="UI" Name="Transfer Syntax
       UID" Length="19">1.2.840.10008.1.2.1</Attribute>
    .....
    <Attribute Tag="00020016" VR="AE" Name="Source
       Application Entity Title"
       Length="15">MERGE_STORE_SCP</Attribute>
  </FileMetaInfo>
  <DataSet Service="STANDARD_SEC_CAPTURE"
       Command="C_STORE_RQ"
       TransferSyntax="1.2.840.10008.1.2.1">
    <Attribute Tag="00080008" VR="CS" Name="Image Type"
       Length="24">ORIGINAL\SECONDARY\OTHER</Attribute>
    <Attribute Tag="00080016" VR="UI" Name="SOP Class UID"
       Length="25">1.2.840.10008.5.1.4.1.1.7</Attribute>
    .....
    <Attribute Tag="00080020" VR="DA" Name="Study Date"
       Length="8">20020717</Attribute>
    <Attribute Tag="00080030" VR="TM" Name="Study Time"
       Length="6">123429</Attribute>
    <Attribute Tag="00080060" VR="CS" Name="Modality"
       Length="2">OT</Attribute>
    .....
    <Attribute Tag="00081111" VR="SQ" Name="Referenced
       Performed Procedure Step Sequence" Length="1">
      <Item>
        <Attribute Tag="00081150" VR="UI" Name="Referenced
       SOP Class UID"
       Length="23">1.2.840.10008.3.1.2.3.3</Attribute>
        <Attribute Tag="00081155" VR="UI" Name="Referenced
       SOP Instance UID"
       Length="44">2.16.840.1.113669.4.960070.844.1026926027
       .44</Attribute>
      </Item>
    </Attribute>
    <Attribute Tag="00090010" VR="LO" Name="Private Creator
       Code" PCode="PrivateCode" Length="11">SAMPLE
       PCODE</Attribute>
    <Attribute Tag="00091010" VR="LO" Name="Private"
       PCode="SAMPLE PCODE" Length="6">Value1</Attribute>
```

```
    <Attribute Tag="00091015" VR="UN" Name="Private"
       PCode="SAMPLE PCODE" Length="6">INAgNAEy</Attribute>
    .....
     <Attribute Tag="00100010" VR="PN" Name="Patient's Name"
       Length="28">Last^First</Attribute>
     .....
     <Attribute Tag="7FE00010" VR="OW" Name="Pixel Data"
       Encoding="Base64"
       Length="262144">HQAABgMAAAIHBAM.....</Attribute>
  </DataSet>
</DcmFile>
```

The default encoding of bulks and attributes with VR UN:

```
<?xml version="1.0" encoding="utf-8"?>
<DcmFile>
  <FileMetaInfo Service="STANDARD_SEC_CAPTURE"
       Command="C_STORE_RQ">
    <Attribute Tag="00020001" VR="OB" Name="File Meta
       Information Version" Length="2">00 01</Attribute>
    .....
    <Attribute Tag="00020016" VR="AE" Name="Source
       Application Entity Title"
       Length="15">MERGE_STORE_SCP</Attribute>
  </FileMetaInfo>
  <DataSet Service="STANDARD_SEC_CAPTURE"
       Command="C_STORE_RQ"
       TransferSyntax="1.2.840.10008.1.2.1">
    <Attribute Tag="00080008" VR="CS" Name="Image Type"
       Length="24">ORIGINAL\SECONDARY\OTHER</Attribute>
    <Attribute Tag="00080016" VR="UI" Name="SOP Class UID"
       Length="25">1.2.840.10008.5.1.4.1.1.7</Attribute>
    .....
    <Attribute Tag="00081111" VR="SQ" Name="Referenced
       Performed Procedure Step Sequence" Length="1">
      <Item>
        <Attribute Tag="00081150" VR="UI" Name="Referenced
       SOP Class UID"
       Length="23">1.2.840.10008.3.1.2.3.3</Attribute>
         <Attribute Tag="00081155" VR="UI" Name="Referenced
       SOP Instance UID"
       Length="44">2.16.840.1.113669.4.960070.844.1026926027
       .44</Attribute>
      </Item>
    </Attribute>
   <Attribute Tag="00090010" VR="LO" Name="Private Creator
       Code" PCode="PrivateCode" Length="11">SAMPLE
       PCODE</Attribute>
    <Attribute Tag="00091010" VR="LO" Name="Private"
       PCode="SAMPLE PCODE" Length="6">Value1</Attribute>
   <Attribute Tag="00091015" VR="UN" Name="Private"
       PCode="SAMPLE PCODE" Length="6">20 20 20 20 20
       30y</Attribute>
    .....
    <Attribute Tag="7FE00010" VR="OW" Name="Pixel Data"
       Encoding="Base64" Length="262144">06 00 04 00 04 00
       02 00 03.....</Attribute>
  </DataSet>
</DcmFile>
```